Lab 4 Report:

Surpass Human Performance in Fashion MNIST Classificaion

Name:

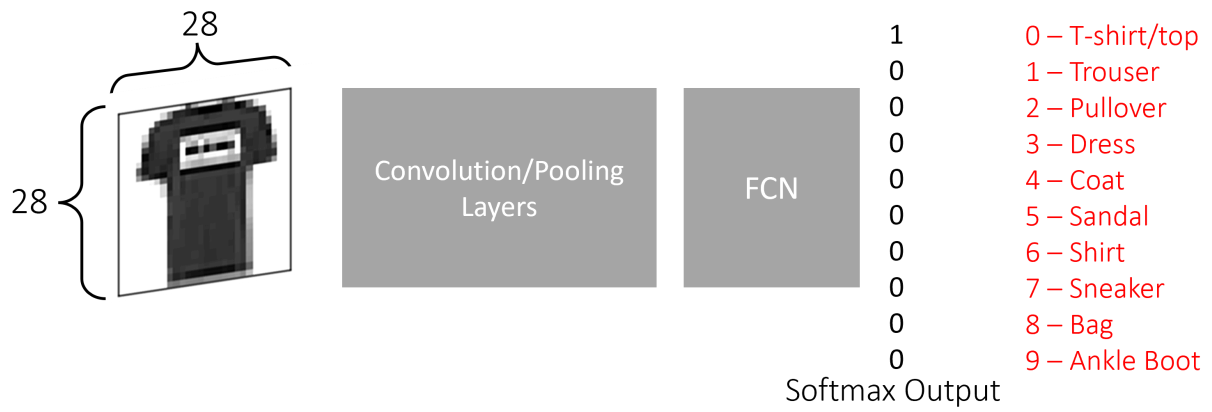
In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import torch
import numpy as np
```

In [2]:
```
from IPython.display import Image # For displaying images in colab jupyter cell
```

In [3]:
```
Image('lab4_exercise.png', width = 1000)
```

Out[3]:

Surpass Human Performance in Fashion MNIST Classification



In this exercise, you will classify fashion item class (28 x 28) using your own **Convolutional Neural Network Architecture.**

Prior to training your neural net, 1) Normalize the dataset using standard scaler and 2) Split the dataset into train/validation/test.

Design your own CNN architecture with your choice of Convolution/Pooling/FCN layers, activation functions, optimization method etc.

Your goal is to **achieve a testing accuracy of >89%**, with no restrictions on epochs **(Human performance: 83.5%).**

Demonstrate the performance of your model via plotting the **training loss, validation accuracy** and printing out the **testing accuracy.**

After your model has reached the goal, print the accuracy in each class. What is the class that your model performed the worst? 44

Prepare Data

In [4]:
```
# Load Fashion-MNIST Dataset in Numpy

# 10000 training features/targets where each feature is a greyscale image with shape (28, 28)
train_features = np.load('fashion_mnist_train_features.npy')
train_targets = np.load('fashion_mnist_train_targets.npy')

# 1000 testing features/targets
```

```
test_features = np.load('fashion_mnist_test_features.npy')
test_targets = np.load('fashion_mnist_test_targets.npy')

# Let's see the shapes of training/testing datasets
print("Training Features Shape: ", train_features.shape)
print("Training Targets Shape: ", train_targets.shape)
print("Testing Features Shape: ", test_features.shape)
print("Testing Targets Shape: ", test_targets.shape)
```

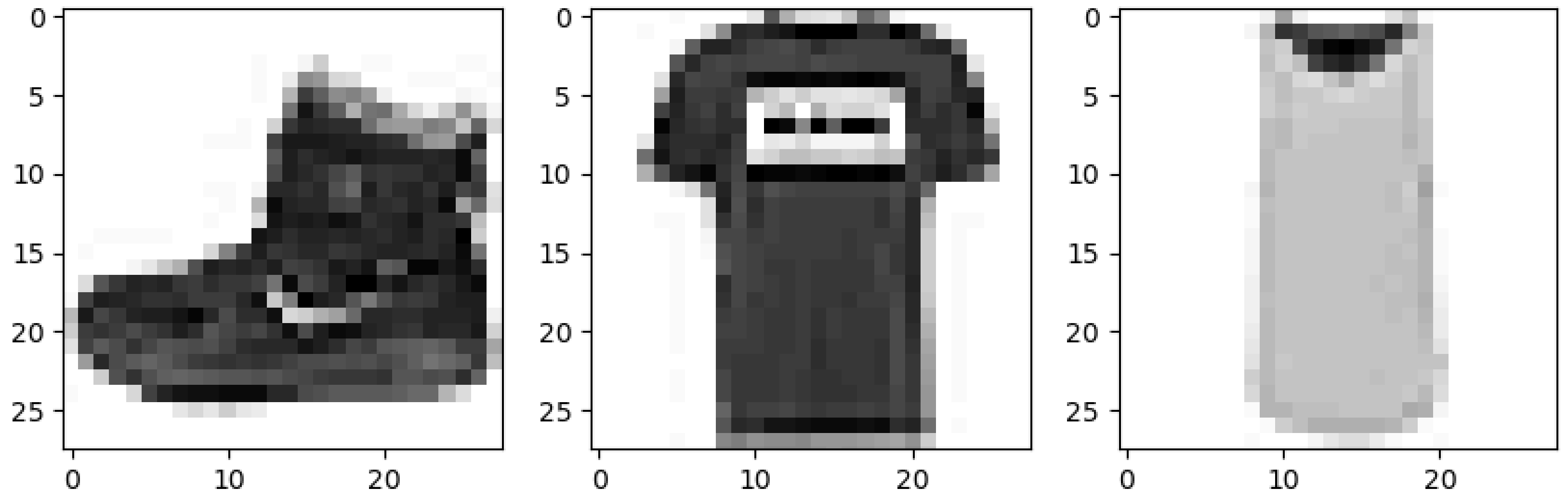
```
Training Features Shape:  (10000, 28, 28)
Training Targets Shape:  (10000,)
Testing Features Shape:  (1000, 28, 28)
Testing Targets Shape:  (1000,)
```

In [5]:

```
# Visualizing the first three training features (samples)

plt.figure(figsize = (10, 10))

plt.subplot(1,3,1)
plt.imshow(train_features[0], cmap = 'Greys')

plt.subplot(1,3,2)
plt.imshow(train_features[1], cmap = 'Greys')

plt.subplot(1,3,3)
plt.imshow(train_features[2], cmap = 'Greys')
```

Out[5]:

```
<matplotlib.image.AxesImage at 0x21ae45bf520>
```



In [6]:
```
# Reshape features via flattening the images
# This refers to reshape each sample from a 2d array to a 1d array.
# hint: np.reshape() function could be useful here
from sklearn.preprocessing import StandardScaler
import torch, torch.nn as nn

tr_flat = train_features.reshape(len(train_features), -1)
te_flat = test_features.reshape(len(test_features),  -1)
scaler  = StandardScaler()
```

In [7]:
```
# Define your scaling function
# YOUR CODE HERE

# Scale the dataset according to standard scaling
train_features = scaler.fit_transform(tr_flat).reshape(-1,28,28)
test_features  = scaler.transform(te_flat).reshape(-1,28,28)
```

In [8]:
```
# Take the first 1000 (or randomly select 1000) training features and targets as validation set

val_features, val_targets   = train_features[:1000], train_targets[:1000]

# Take the remaining 9000 training features and targets as training set
```

```
train_features, train_targets = train_features[1000:], train_targets[1000:]
```

In [9]:
```
# Reshape train/validation/test sets to conform to PyTorch's (N, Channels, Height, Width) standard for CNNs

train_features = train_features[:, np.newaxis, :, :]      # (N,1,28,28)
val_features   = val_features[:,   np.newaxis, :, :]      # (N_val,1,28,28)
test_features  = test_features[:,  np.newaxis, :, :]      # (N_test,1,28,28)

print("Shapes after add channel axis:",
      train_features.shape, val_features.shape, test_features.shape)
```

```
Shapes after add channel axis: (9000, 1, 28, 28) (1000, 1, 28, 28) (1000, 1, 28, 28)
```

Define Model

In [10]:
```
# Define your CNN architecture here

class CNNModel(nn.Module):
    """
    Two-conv modern CNN for Fashion-MNIST.
    Keeps the SAME spatial sizes as the course example (28→14→7) so the
    flattened feature vector is 32 × 7 × 7 = 1568.

    Layer order:
      Conv3×3 → BN → ReLU → MaxPool
      Conv3×3 → BN → ReLU → MaxPool
      Flatten  → Dropout → FC-10 (logits)
    """
    def __init__(self):
        super().__init__()

        # ---- Block 1 ---------------------------------------------------
        # Input : (N, 1, 28, 28)
        # Output: (N, 16, 28, 28)   – same H/W thanks to padding=1
        self.conv1 = nn.Conv2d(in_channels=1,  out_channels=16,
                               kernel_size=3,  padding=1)
        self.bn1   = nn.BatchNorm2d(16)         # stabilise activations
        self.pool1 = nn.MaxPool2d(kernel_size=2) # 28 → 14
        # ---- Block 2 ----------------------------------------------------
        # Input : (N, 16, 14, 14)
        # Output: (N, 32, 14, 14)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32,
                               kernel_size=3, padding=1)
        self.bn2   = nn.BatchNorm2d(32)
        self.pool2 = nn.MaxPool2d(kernel_size=2) # 14 → 7

        # ---- Classifier -------------------------------------------------
        # Flatten: (N, 32, 7, 7) → (N, 1568)
        self.dropout = nn.Dropout(p=0.4)         # reduce over-fitting
        self.fc1     = nn.Linear(32 * 7 * 7, 10) # 10 logits for CE loss

    def forward(self, x):
        """
        Forward pass:
          x : Tensor of shape (batch, 1, 28, 28)
          returns raw logits (no soft-max; CrossEntropyLoss handles it)
        """
        # Block 1
        x = self.conv1(x)        # convolution
        x = self.bn1(x)          # batch normalisation
        x = torch.relu(x)        # non-linearity
        x = self.pool1(x)        # spatial down-sampling → 14×14

        # Block 2
        x = self.conv2(x)
        x = self.bn2(x)
        x = torch.relu(x)
        x = self.pool2(x)        # → 7×7

        # Classifier
        x = x.view(x.size(0), -1)  # flatten to (N, 1568)
        x = self.dropout(x)
        logits = self.fc1(x)

        return logits
```

Select Hyperparameters

In [11]:
```
# Fix the random seed so that model performance is reproducible

DEVICE = 'cuda' if torch.cuda.is_available() else 'cpu'
print("Using device:", DEVICE)
torch.manual_seed(55)

# Initialize your CNN model

model = CNNModel().to(DEVICE)

# Define learning rate, epoch and batchsize for mini-batch gradient

learning_rate = 1e-4       # Adam → 1 × 10⁻4
epochs        = 250
batchsize     = 128


# Define loss function and optimizer

loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

model
```

```
Using device: cuda
```

Out[11]:
```
CNNModel(
  (conv1): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout): Dropout(p=0.4, inplace=False)
  (fc1): Linear(in_features=1568, out_features=10, bias=True)
)
```

Identify Tracked Values

In [12]:
```
# Placeholders for training loss and validation accuracy during training
# Training loss should be tracked for each iteration (1 iteration -> single forward pass to the network)
# Validation accuracy should be evaluated every 'Epoch' (1 epoch -> full training dataset)
# If using batch gradient, 1 iteration = 1 epoch

train_loss_list          = []
val_accuracy_list        = np.zeros(epochs)
```

Train Model

In [13]:
```
import tqdm # Use "for epoch in tqdm.trange(epochs):" to see the progress bar

# Convert the training, validation, testing dataset (NumPy arrays) into torch tensors
# Split your training features/targets into mini-batches if using mini-batch gradient

# ------------------------------------------------------------------
# Convert NumPy → Torch tensors, move to DEVICE (CPU or GPU)
# ------------------------------------------------------------------
tr_x = torch.from_numpy(train_features).float().to(DEVICE)   # training images
tr_y = torch.from_numpy(train_targets).long().to(DEVICE)     # training labels
val_x = torch.from_numpy(val_features).float().to(DEVICE)    # validation images
val_y = torch.from_numpy(val_targets).long().to(DEVICE)      # validation labels
te_x  = torch.from_numpy(test_features).float().to(DEVICE)   # test images
te_y  = torch.from_numpy(test_targets).long().to(DEVICE)     # test labels

# Split the training set into mini-batches (128 samples each here)
tr_batches_x = torch.split(tr_x, batchsize)
tr_batches_y = torch.split(tr_y, batchsize)

# ------------------------------------------------------------------
# Training loop
# ------------------------------------------------------------------
print("Model on:", next(model.parameters()).device)  # sanity-check: should print cuda:0

for epoch in tqdm.trange(epochs, desc='Epoch'):
```

```python
    # ------ train mode ------
    model.train()
    for xb, yb in zip(tr_batches_x, tr_batches_y):
        optimizer.zero_grad()                # reset accumulated gradients
        logits = model(xb)                   # forward pass
        loss   = loss_func(logits, yb)       # compute cross-entropy
        train_loss_list.append(loss.item()) # stash for plotting
        loss.backward()                      # back-propagate
        optimizer.step()                     # update weights

    # ------ validation ------
    model.eval()                             # turn off dropout / BN updates
    with torch.no_grad():                    # no grad tracking ⇒ lower memory
        preds = model(val_x).argmax(dim=1)  # get predicted class indices
        acc   = (preds == val_y).float().mean().item()
        val_accuracy_list[epoch] = acc

    # Console feedback per epoch
    print(f"Epoch {epoch:02d}  Val Acc = {acc*100:5.2f}%")
```

```
Model on: cuda:0
Epoch:   1%|          | 2/250 [00:00<00:46,  5.37it/s]
Epoch 00  Val Acc = 72.10%
Epoch 01  Val Acc = 75.30%
Epoch:   1%|          | 3/250 [00:00<00:36,  6.76it/s]
Epoch 02  Val Acc = 77.80%
Epoch 03  Val Acc = 79.70%
Epoch:   2%|▏         | 6/250 [00:00<00:29,  8.25it/s]
Epoch 04  Val Acc = 80.80%
Epoch 05  Val Acc = 82.20%
Epoch:   4%|▎         | 9/250 [00:01<00:25,  9.46it/s]
Epoch 06  Val Acc = 82.90%
Epoch 07  Val Acc = 83.50%
Epoch 08  Val Acc = 84.50%
Epoch:   5%|▍         | 12/250 [00:01<00:22, 10.47it/s]
Epoch 09  Val Acc = 84.70%
Epoch 10  Val Acc = 85.10%
Epoch 11  Val Acc = 85.30%
Epoch:   6%|▌         | 14/250 [00:01<00:22, 10.36it/s]
```

```
Epoch 12  Val Acc = 85.70%
Epoch 13  Val Acc = 85.60%
Epoch:   6%|█         | 16/250 [00:01<00:23, 10.17it/s]
Epoch 14  Val Acc = 86.10%
Epoch 15  Val Acc = 86.60%
Epoch 16  Val Acc = 86.60%
Epoch:   8%|█         | 20/250 [00:02<00:21, 10.86it/s]
Epoch 17  Val Acc = 86.80%
Epoch 18  Val Acc = 87.00%
Epoch 19  Val Acc = 87.00%
Epoch:   9%|█         | 22/250 [00:02<00:20, 11.25it/s]
Epoch 20  Val Acc = 87.20%
Epoch 21  Val Acc = 87.10%
Epoch 22  Val Acc = 87.50%
Epoch:  10%|█         | 24/250 [00:02<00:19, 11.72it/s]
Epoch 23  Val Acc = 87.30%
Epoch 24  Val Acc = 87.10%
Epoch:  10%|█        | 26/250 [00:02<00:19, 11.21it/s]
Epoch 25  Val Acc = 87.30%
Epoch 26  Val Acc = 87.70%
Epoch:  12%|█▏        | 30/250 [00:03<00:20, 10.61it/s]
Epoch 27  Val Acc = 87.60%
Epoch 28  Val Acc = 87.90%
Epoch 29  Val Acc = 88.50%
Epoch:  13%|█▎        | 32/250 [00:03<00:19, 10.97it/s]
Epoch 30  Val Acc = 88.20%
Epoch 31  Val Acc = 88.40%
Epoch 32  Val Acc = 88.50%
Epoch:  14%|█▍        | 36/250 [00:03<00:19, 11.22it/s]
Epoch 33  Val Acc = 88.30%
Epoch 34  Val Acc = 88.50%
Epoch 35  Val Acc = 88.90%
Epoch:  15%|█        | 38/250 [00:03<00:19, 10.74it/s]
Epoch 36  Val Acc = 88.60%
Epoch 37  Val Acc = 88.70%
Epoch 38  Val Acc = 89.00%
Epoch:  17%|██        | 42/250 [00:04<00:19, 10.93it/s]
```

```
Epoch 39  Val Acc = 89.10%
Epoch 40  Val Acc = 88.90%
Epoch 41  Val Acc = 88.80%
Epoch:  18%|█▊        | 44/250 [00:04<00:18, 11.28it/s]
Epoch 42  Val Acc = 89.00%
Epoch 43  Val Acc = 88.70%
Epoch 44  Val Acc = 89.00%
Epoch:  19%|█▉        | 48/250 [00:04<00:17, 11.41it/s]
Epoch 45  Val Acc = 89.10%
Epoch 46  Val Acc = 88.80%
Epoch 47  Val Acc = 89.20%
Epoch:  20%|██        | 50/250 [00:04<00:19, 10.29it/s]
Epoch 48  Val Acc = 89.20%
Epoch 49  Val Acc = 89.00%
Epoch:  21%|██        | 52/250 [00:05<00:18, 10.45it/s]
Epoch 50  Val Acc = 89.20%
Epoch 51  Val Acc = 88.80%
Epoch 52  Val Acc = 89.00%
Epoch:  22%|██▏       | 56/250 [00:05<00:17, 11.27it/s]
Epoch 53  Val Acc = 88.80%
Epoch 54  Val Acc = 88.80%
Epoch 55  Val Acc = 89.30%
Epoch:  23%|██▎       | 58/250 [00:05<00:16, 11.56it/s]
Epoch 56  Val Acc = 89.10%
Epoch 57  Val Acc = 89.00%
Epoch 58  Val Acc = 88.90%
Epoch:  25%|██▍       | 62/250 [00:05<00:17, 10.69it/s]
Epoch 59  Val Acc = 88.90%
Epoch 60  Val Acc = 89.20%
Epoch 61  Val Acc = 89.20%
Epoch:  26%|██▌       | 64/250 [00:06<00:18, 10.30it/s]
Epoch 62  Val Acc = 89.10%
Epoch 63  Val Acc = 89.00%
Epoch:  26%|██▋       | 66/250 [00:06<00:18, 10.03it/s]
Epoch 64  Val Acc = 89.20%
Epoch 65  Val Acc = 89.60%
Epoch 66  Val Acc = 89.40%
Epoch:  27%|██▋       | 68/250 [00:06<00:18,  9.93it/s]
```

```
Epoch 67  Val Acc = 89.40%
Epoch 68  Val Acc = 89.20%
Epoch:  28%|██▊       | 71/250 [00:06<00:18,  9.54it/s]
Epoch 69  Val Acc = 89.40%
Epoch 70  Val Acc = 89.20%
Epoch 71  Val Acc = 89.20%
Epoch:  30%|███       | 75/250 [00:07<00:16, 10.89it/s]
Epoch 72  Val Acc = 89.30%
Epoch 73  Val Acc = 89.90%
Epoch 74  Val Acc = 89.60%
Epoch:  31%|███       | 77/250 [00:07<00:16, 10.66it/s]
Epoch 75  Val Acc = 89.60%
Epoch 76  Val Acc = 89.50%
Epoch:  32%|███▏      | 79/250 [00:07<00:16, 10.27it/s]
Epoch 77  Val Acc = 89.50%
Epoch 78  Val Acc = 89.50%
Epoch:  32%|███▏      | 81/250 [00:07<00:16, 10.00it/s]
Epoch 79  Val Acc = 89.70%
Epoch 80  Val Acc = 89.50%
Epoch:  33%|███▎      | 83/250 [00:08<00:16,  9.95it/s]
Epoch 81  Val Acc = 89.40%
Epoch 82  Val Acc = 89.60%
Epoch:  34%|███▍      | 85/250 [00:08<00:17,  9.70it/s]
Epoch 83  Val Acc = 89.50%
Epoch 84  Val Acc = 89.50%
Epoch 85  Val Acc = 89.60%
Epoch:  36%|███▌      | 89/250 [00:08<00:15, 10.35it/s]
Epoch 86  Val Acc = 89.80%
Epoch 87  Val Acc = 89.60%
Epoch 88  Val Acc = 89.80%
Epoch:  36%|███▋      | 91/250 [00:08<00:15, 10.05it/s]
Epoch 89  Val Acc = 89.70%
Epoch 90  Val Acc = 89.40%
Epoch:  37%|███▋      | 93/250 [00:09<00:15, 10.16it/s]
Epoch 91  Val Acc = 89.90%
Epoch 92  Val Acc = 89.80%
Epoch 93  Val Acc = 89.80%
```

```
Epoch:  39%|███▊      | 97/250 [00:09<00:14, 10.91it/s]
Epoch 94  Val Acc = 89.80%
Epoch 95  Val Acc = 89.50%
Epoch 96  Val Acc = 89.70%
Epoch:  40%|███▊      | 99/250 [00:09<00:14, 10.36it/s]
Epoch 97  Val Acc = 89.70%
Epoch 98  Val Acc = 89.50%
Epoch:  40%|████      | 101/250 [00:09<00:14, 10.17it/s]
Epoch 99  Val Acc = 89.60%
Epoch 100  Val Acc = 90.20%
Epoch:  41%|████      | 103/250 [00:10<00:14, 10.35it/s]
Epoch 101  Val Acc = 90.10%
Epoch 102  Val Acc = 89.80%
Epoch 103  Val Acc = 89.90%
Epoch:  43%|████▎     | 107/250 [00:10<00:13, 10.67it/s]
Epoch 104  Val Acc = 90.10%
Epoch 105  Val Acc = 89.60%
Epoch 106  Val Acc = 90.00%
Epoch:  44%|████▎     | 109/250 [00:10<00:13, 10.36it/s]
Epoch 107  Val Acc = 90.00%
Epoch 108  Val Acc = 90.10%
Epoch:  44%|████▍     | 111/250 [00:10<00:14,  9.87it/s]
Epoch 109  Val Acc = 89.90%
Epoch 110  Val Acc = 89.70%
Epoch:  46%|████▌     | 114/250 [00:11<00:13, 10.28it/s]
Epoch 111  Val Acc = 89.50%
Epoch 112  Val Acc = 89.90%
Epoch 113  Val Acc = 90.10%
Epoch:  46%|████▋     | 116/250 [00:11<00:12, 10.83it/s]
Epoch 114  Val Acc = 89.80%
Epoch 115  Val Acc = 89.60%
Epoch 116  Val Acc = 90.00%
Epoch:  48%|████▊     | 120/250 [00:11<00:11, 11.53it/s]
Epoch 117  Val Acc = 89.80%
Epoch 118  Val Acc = 89.80%
Epoch 119  Val Acc = 89.40%
Epoch:  49%|████▉     | 122/250 [00:11<00:12, 10.65it/s]
```

```
Epoch 120  Val Acc = 89.70%
Epoch 121  Val Acc = 89.90%
Epoch:  50%|████▉     | 124/250 [00:12<00:12, 10.12it/s]
Epoch 122  Val Acc = 90.20%
Epoch 123  Val Acc = 90.20%
Epoch:  50%|█████     | 126/250 [00:12<00:12,  9.99it/s]
Epoch 124  Val Acc = 89.70%
Epoch 125  Val Acc = 89.70%
Epoch:  51%|█████     | 128/250 [00:12<00:12,  9.93it/s]
Epoch 126  Val Acc = 89.90%
Epoch 127  Val Acc = 89.60%
Epoch 128  Val Acc = 89.40%
Epoch:  52%|█████▏    | 131/250 [00:12<00:12,  9.52it/s]
Epoch 129  Val Acc = 90.00%
Epoch 130  Val Acc = 90.00%
Epoch:  53%|█████▎    | 133/250 [00:12<00:12,  9.58it/s]
Epoch 131  Val Acc = 90.00%
Epoch 132  Val Acc = 89.90%
Epoch 133  Val Acc = 89.90%
Epoch:  55%|█████▍    | 137/250 [00:13<00:10, 10.56it/s]
Epoch 134  Val Acc = 89.90%
Epoch 135  Val Acc = 90.00%
Epoch 136  Val Acc = 90.00%
Epoch:  56%|█████▌    | 139/250 [00:13<00:10, 10.46it/s]
Epoch 137  Val Acc = 89.70%
Epoch 138  Val Acc = 90.10%
Epoch 139  Val Acc = 89.60%
Epoch:  56%|█████▋    | 141/250 [00:13<00:09, 10.97it/s]
Epoch 140  Val Acc = 90.00%
Epoch 141  Val Acc = 89.80%
Epoch:  58%|█████▊    | 145/250 [00:14<00:10, 10.47it/s]
Epoch 142  Val Acc = 90.10%
Epoch 143  Val Acc = 90.10%
Epoch 144  Val Acc = 89.80%
Epoch:  59%|█████▉    | 147/250 [00:14<00:09, 10.56it/s]
Epoch 145  Val Acc = 90.10%
Epoch 146  Val Acc = 90.00%
Epoch 147  Val Acc = 89.90%
```

```
Epoch:  60%|██████    | 151/250 [00:14<00:09, 10.37it/s]
Epoch 148  Val Acc = 90.10%
Epoch 149  Val Acc = 90.00%
Epoch 150  Val Acc = 90.00%
Epoch:  61%|██████    | 153/250 [00:14<00:09,  9.74it/s]
Epoch 151  Val Acc = 89.70%
Epoch 152  Val Acc = 90.20%
Epoch:  62%|██████▏   | 156/250 [00:15<00:09, 10.38it/s]
Epoch 153  Val Acc = 90.10%
Epoch 154  Val Acc = 90.50%
Epoch 155  Val Acc = 90.60%
Epoch:  63%|██████▎   | 158/250 [00:15<00:08, 10.97it/s]
Epoch 156  Val Acc = 90.10%
Epoch 157  Val Acc = 89.90%
Epoch 158  Val Acc = 90.30%
Epoch:  64%|██████▍   | 160/250 [00:15<00:08, 10.71it/s]
Epoch 159  Val Acc = 90.40%
Epoch 160  Val Acc = 90.10%
Epoch:  66%|██████▌   | 164/250 [00:15<00:08, 10.22it/s]
Epoch 161  Val Acc = 90.10%
Epoch 162  Val Acc = 90.40%
Epoch 163  Val Acc = 90.00%
Epoch:  66%|██████▋   | 166/250 [00:16<00:07, 10.54it/s]
Epoch 164  Val Acc = 89.90%
Epoch 165  Val Acc = 90.70%
Epoch 166  Val Acc = 90.20%
Epoch:  67%|██████▋   | 168/250 [00:16<00:07, 10.61it/s]
Epoch 167  Val Acc = 90.30%
Epoch 168  Val Acc = 90.60%
Epoch:  69%|██████▉   | 172/250 [00:16<00:07, 10.98it/s]
Epoch 169  Val Acc = 90.00%
Epoch 170  Val Acc = 89.80%
Epoch 171  Val Acc = 90.00%
Epoch:  70%|██████▉   | 174/250 [00:16<00:07, 10.36it/s]
Epoch 172  Val Acc = 90.10%
Epoch 173  Val Acc = 90.40%
Epoch:  70%|███████   | 176/250 [00:17<00:07, 10.37it/s]
```

```
Epoch 174  Val Acc = 90.10%
Epoch 175  Val Acc = 89.90%
Epoch 176  Val Acc = 90.40%
Epoch:  72%|███████▏  | 180/250 [00:17<00:06, 11.29it/s]
Epoch 177  Val Acc = 90.40%
Epoch 178  Val Acc = 90.10%
Epoch 179  Val Acc = 89.90%
Epoch:  73%|███████▎  | 182/250 [00:17<00:05, 11.54it/s]
Epoch 180  Val Acc = 90.20%
Epoch 181  Val Acc = 90.00%
Epoch 182  Val Acc = 90.30%
Epoch:  74%|███████▍  | 186/250 [00:17<00:05, 11.02it/s]
Epoch 183  Val Acc = 90.10%
Epoch 184  Val Acc = 90.10%
Epoch 185  Val Acc = 90.40%
Epoch:  75%|███████▌  | 188/250 [00:18<00:05, 11.14it/s]
Epoch 186  Val Acc = 89.90%
Epoch 187  Val Acc = 90.20%
Epoch 188  Val Acc = 90.10%
Epoch:  77%|███████▋  | 192/250 [00:18<00:04, 11.81it/s]
Epoch 189  Val Acc = 90.00%
Epoch 190  Val Acc = 89.90%
Epoch 191  Val Acc = 90.30%
Epoch:  78%|███████▊  | 194/250 [00:18<00:04, 12.06it/s]
Epoch 192  Val Acc = 90.00%
Epoch 193  Val Acc = 90.00%
Epoch 194  Val Acc = 90.20%
Epoch:  78%|███████▊  | 196/250 [00:18<00:04, 11.85it/s]
Epoch 195  Val Acc = 90.00%
Epoch 196  Val Acc = 90.20%
Epoch:  80%|████████  | 200/250 [00:19<00:04, 11.25it/s]
Epoch 197  Val Acc = 89.90%
Epoch 198  Val Acc = 89.70%
Epoch 199  Val Acc = 89.50%
Epoch:  81%|████████  | 202/250 [00:19<00:04, 11.65it/s]
Epoch 200  Val Acc = 90.00%
Epoch 201  Val Acc = 90.20%
Epoch 202  Val Acc = 89.90%
```

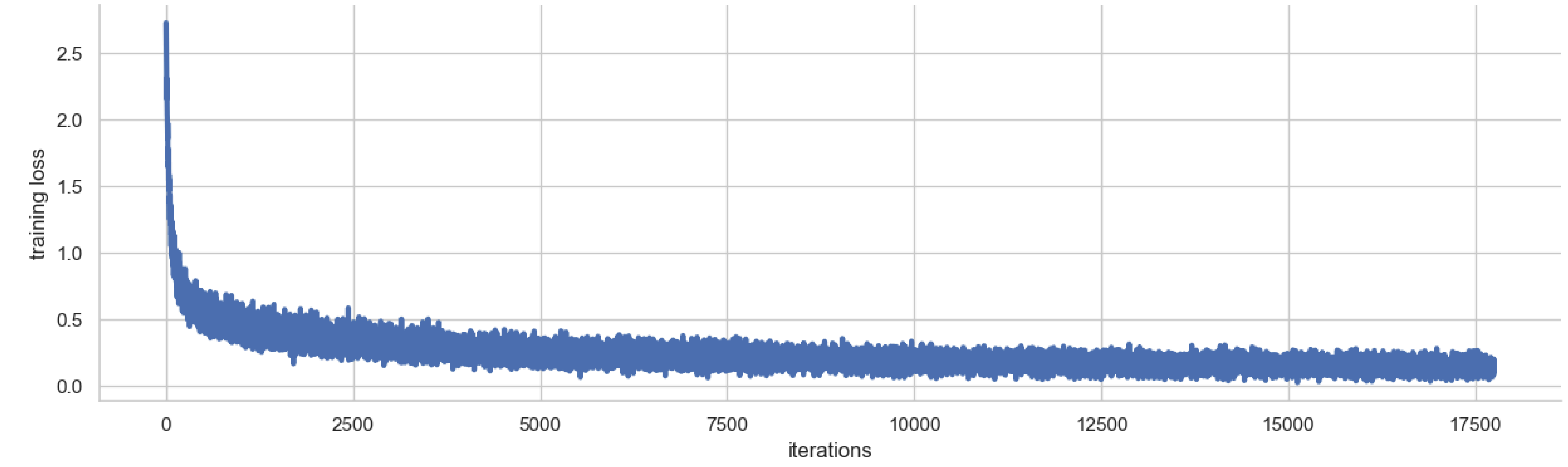
```
Epoch:  82%|████████▏ | 204/250 [00:19<00:04, 11.20it/s]
Epoch 203  Val Acc = 89.70%
Epoch 204  Val Acc = 90.10%
Epoch:  82%|████████▏ | 206/250 [00:19<00:04, 10.60it/s]
Epoch 205  Val Acc = 90.00%
Epoch 206  Val Acc = 90.10%
Epoch:  84%|████████▍ | 210/250 [00:20<00:03, 10.71it/s]
Epoch 207  Val Acc = 90.20%
Epoch 208  Val Acc = 90.00%
Epoch 209  Val Acc = 90.10%
Epoch:  85%|████████▍ | 212/250 [00:20<00:03, 10.95it/s]
Epoch 210  Val Acc = 89.90%
Epoch 211  Val Acc = 90.10%
Epoch 212  Val Acc = 89.90%
Epoch:  86%|████████▋ | 216/250 [00:20<00:03, 11.16it/s]
Epoch 213  Val Acc = 89.90%
Epoch 214  Val Acc = 90.20%
Epoch 215  Val Acc = 89.90%
Epoch:  87%|████████▋ | 218/250 [00:20<00:02, 10.86it/s]
Epoch 216  Val Acc = 90.00%
Epoch 217  Val Acc = 90.10%
Epoch:  88%|████████▊ | 220/250 [00:20<00:02, 10.37it/s]
Epoch 218  Val Acc = 89.90%
Epoch 219  Val Acc = 89.80%
Epoch 220  Val Acc = 89.70%
Epoch:  90%|████████▉ | 224/250 [00:21<00:02, 11.23it/s]
Epoch 221  Val Acc = 90.00%
Epoch 222  Val Acc = 90.10%
Epoch 223  Val Acc = 89.70%
Epoch:  90%|█████████ | 226/250 [00:21<00:02, 11.63it/s]
Epoch 224  Val Acc = 90.00%
Epoch 225  Val Acc = 90.00%
Epoch 226  Val Acc = 89.50%
Epoch:  92%|█████████▏| 230/250 [00:21<00:01, 11.95it/s]
Epoch 227  Val Acc = 90.30%
Epoch 228  Val Acc = 89.70%
Epoch 229  Val Acc = 90.10%
```

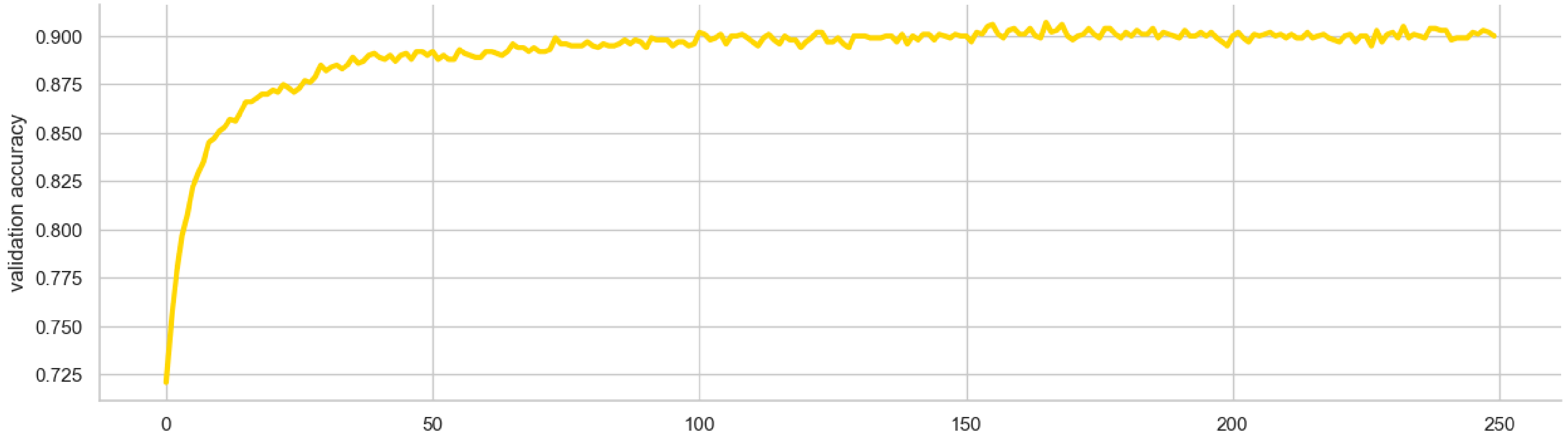
```
Epoch:  93%|█████████▎| 232/250 [00:21<00:01, 12.22it/s]
Epoch 230  Val Acc = 90.20%
Epoch 231  Val Acc = 89.90%
Epoch 232  Val Acc = 90.50%
Epoch:  94%|█████████▍| 236/250 [00:22<00:01, 12.16it/s]
Epoch 233  Val Acc = 89.90%
Epoch 234  Val Acc = 90.10%
Epoch 235  Val Acc = 90.00%
Epoch:  95%|█████████▌| 238/250 [00:22<00:00, 12.18it/s]
Epoch 236  Val Acc = 89.90%
Epoch 237  Val Acc = 90.40%
Epoch 238  Val Acc = 90.40%
Epoch:  97%|█████████▋| 242/250 [00:22<00:00, 11.52it/s]
Epoch 239  Val Acc = 90.30%
Epoch 240  Val Acc = 90.30%
Epoch 241  Val Acc = 89.80%
Epoch:  98%|█████████▊| 244/250 [00:23<00:00, 10.94it/s]
Epoch 242  Val Acc = 89.90%
Epoch 243  Val Acc = 89.90%
Epoch 244  Val Acc = 89.90%
Epoch:  99%|█████████▉| 248/250 [00:23<00:00, 11.75it/s]
Epoch 245  Val Acc = 90.20%
Epoch 246  Val Acc = 90.10%
Epoch 247  Val Acc = 90.30%
Epoch: 100%|██████████| 250/250 [00:23<00:00, 10.64it/s]
Epoch 248  Val Acc = 90.20%
Epoch 249  Val Acc = 90.00%
```

Visualize & Evaluate Model

In [14]:
```python
# Seaborn for prettier plot

import seaborn as sns

sns.set(style = 'whitegrid', font_scale = 1)
```

In [15]:
```
# Visualize training loss

plt.figure(figsize = (15, 9))

plt.subplot(2, 1, 1)
plt.plot(train_loss_list, linewidth = 3)
plt.ylabel("training loss")
plt.xlabel("iterations")
sns.despine()

plt.subplot(2, 1, 2)
plt.plot(val_accuracy_list, linewidth = 3, color = 'gold')
plt.ylabel("validation accuracy")
sns.despine()
```





We found that the learning curve when learning rate is e^-3 is spiking, which may be due to the poor choice of learning rate.

After changing it to e^-4, the curve is smoother

In [16]:
```python
# Compute the testing accuracy

# YOUR CODE HERE
# ------------------------------------------------------------------
#   Evaluate the final model on the *held-out* test set
# ------------------------------------------------------------------
model.eval()                                # inference mode: no dropout/B-N updates
with torch.no_grad():                       # disable autograd → lower memory
    logits     = model(te_x)                # forward pass on all test images
    test_preds = logits.argmax(dim=1)       # predicted class indices
    test_acc   = (test_preds == te_y).float().mean().item()

print(f"\nOverall Test Accuracy: {test_acc*100:.2f}%")

# ------------------------------------------------------------------
#   Per-class accuracy analysis
# ------------------------------------------------------------------
labels = [
    'T-shirt', 'Trouser', 'Pullover', 'Dress', 'Coat',
    'Sandal',  'Shirt',   'Sneaker',  'Bag',   'AnkleBoot'
]

cls_corr = [0] * 10        # correct predictions per class
cls_tot  = [0] * 10        # total samples per class

# Tally results class-by-class
for true_lbl, pred_lbl in zip(te_y, test_preds):
    cls_tot[true_lbl]  += 1
    cls_corr[true_lbl] += int(pred_lbl == true_lbl)
```

```
Overall Test Accuracy: 89.20%
```

In [17]:
```python
# (OPTIONAL) Print the testing accuracy for each fashion class. Your code should produce something that looks like:
# Clever usage of np.where() could be useful here

# "Accuracy of T-shirt/top: 93.5 %"
# "Accuracy of Trouser: 89.3 %"
# etc...

# What's the fashion item that your model had the hardest time classifying?
```

```python
accuracies = [c/t for c, t in zip(cls_corr, cls_tot)]
worst = int(np.argmin(accuracies))
# YOUR CODE HERE
for i,lbl in enumerate(labels):
    print(f"{lbl:10s}: {cls_corr[i]/cls_tot[i]*100:6.2f}%")
print(f"\nWorst-classified item: {labels[worst]}")
```

```
T-shirt   :  88.79%
Trouser   :  99.05%
Pullover  :  85.59%
Dress     :  83.87%
Coat      :  80.87%
Sandal    :  95.40%
Shirt     :  73.20%
Sneaker   :  95.79%
Bag       :  96.84%
AnkleBoot :  94.74%

Worst-classified item: Shirt
```