Fashion-MNIST

GEL521 - Machine Learning
202420

PRESENTED TO
Dr Hayssam SERHAN

PRESENTED BY
Antonio HADDAD 202200238
Elias-Charbel SALAMEH 202201047

Agenda

Preparation

After learning how to use the Artificial Neural Network for various endpoints, we aim to operate on a Fashion-MNIST dataset.

Back to Agenda

Back to Agenda

Proposed Solutions

Solution # 1

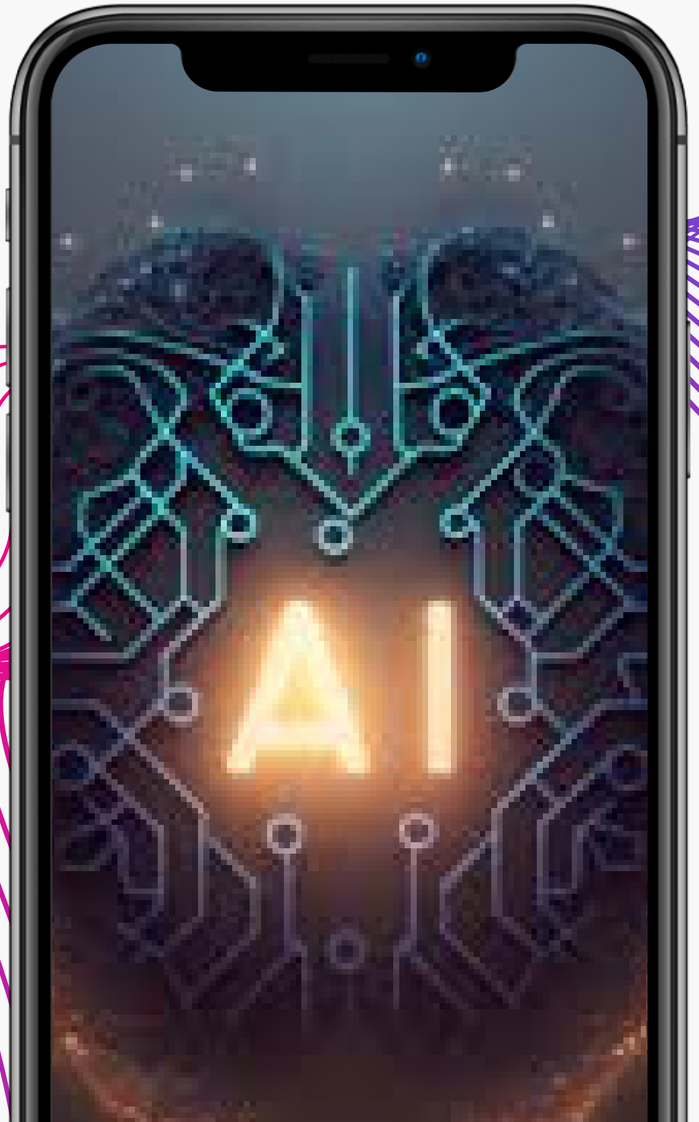
Using the MLPClassifier

Solution # 2

Using the Convolutional Neural Network

Solution # 3

Using our own developed code



Objectives

T = P

around 90%

MATCH THE TRUE OUTPUT
Using a CNN, we need to avoid having mismatches between predicted and true values

IMPROVE THE ACCURACY
Using the ANN increase the accuracy as much as possible

Back to Agenda

Labels

Each training and test example is assigned to one of the following labels:

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Back to Agenda

This is the desired output for the fashion mnist dataset.

Using an MLP Multi-Layer Perceptron Classifier library

```python
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
import numpy as np

# Load MNIST data from openml
fashion_mnist = fetch_openml('Fashion-MNIST') # mnist_784
X = fashion_mnist.data / 255.0  # Scale pixel values to [0, 1]
y = np.array(fashion_mnist.target, dtype=int)
```

```
The number of features is:  784
The number of samples is:  70000
The dimension of X is:  (70000, 784)
```

```python
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create MLPClassifier model with desired hyperparameters
model = MLPClassifier(hidden_layer_sizes=(512,512,128,), activation='relu', solver='adam', max_iter=300, verbose=True)

# Train the model on the training data
model.fit(X_train, y_train)

# Evaluate the model on the testing data
accuracy_train = model.score(X_train,y_train)
accuracy_test = model.score(X_test, y_test)
print("Train set accuracy: {:.2f}".format(accuracy_train * 100))
print("Test set accuracy: {:.2f}".format(accuracy_test * 100))
```

Back to Agenda

```
Iteration 73, loss = 0.03449245
Iteration 74, loss = 0.03790863
Iteration 75, loss = 0.03711825
Iteration 76, loss = 0.02952226
Iteration 77, loss = 0.02931881
Iteration 78, loss = 0.03665593
Iteration 79, loss = 0.03027270
Iteration 80, loss = 0.02968127
Iteration 81, loss = 0.02659590
Iteration 82, loss = 0.03948406
Iteration 83, loss = 0.02727297
Iteration 84, loss = 0.03260940
Iteration 85, loss = 0.02836959
Iteration 86, loss = 0.02843672
Iteration 87, loss = 0.02989809
Iteration 88, loss = 0.01780911
Iteration 89, loss = 0.03861433
Iteration 90, loss = 0.02471355
Iteration 91, loss = 0.01968715
Iteration 92, loss = 0.03941532
Iteration 93, loss = 0.01912692
Iteration 94, loss = 0.02992083
Iteration 95, loss = 0.02960254
Iteration 96, loss = 0.01979922
Iteration 97, loss = 0.03463665
Iteration 98, loss = 0.02079060
Iteration 99, loss = 0.02968828
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Train set accuracy: 99.36
Test set accuracy: 90.14
```

Back to Agenda

Using the keras CNN Convolutional Neural Network

```python
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from tensorflow import keras
from keras import layers

# Load MNIST dataset
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

# Preprocess input images
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255.0
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255.0

# Define model architecture
model = keras.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10)
])

# Compile model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Train model
model.fit(train_images, train_labels, epochs=30, batch_size=128)

# Evaluate model
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f'Test accuracy: {test_acc}')
```

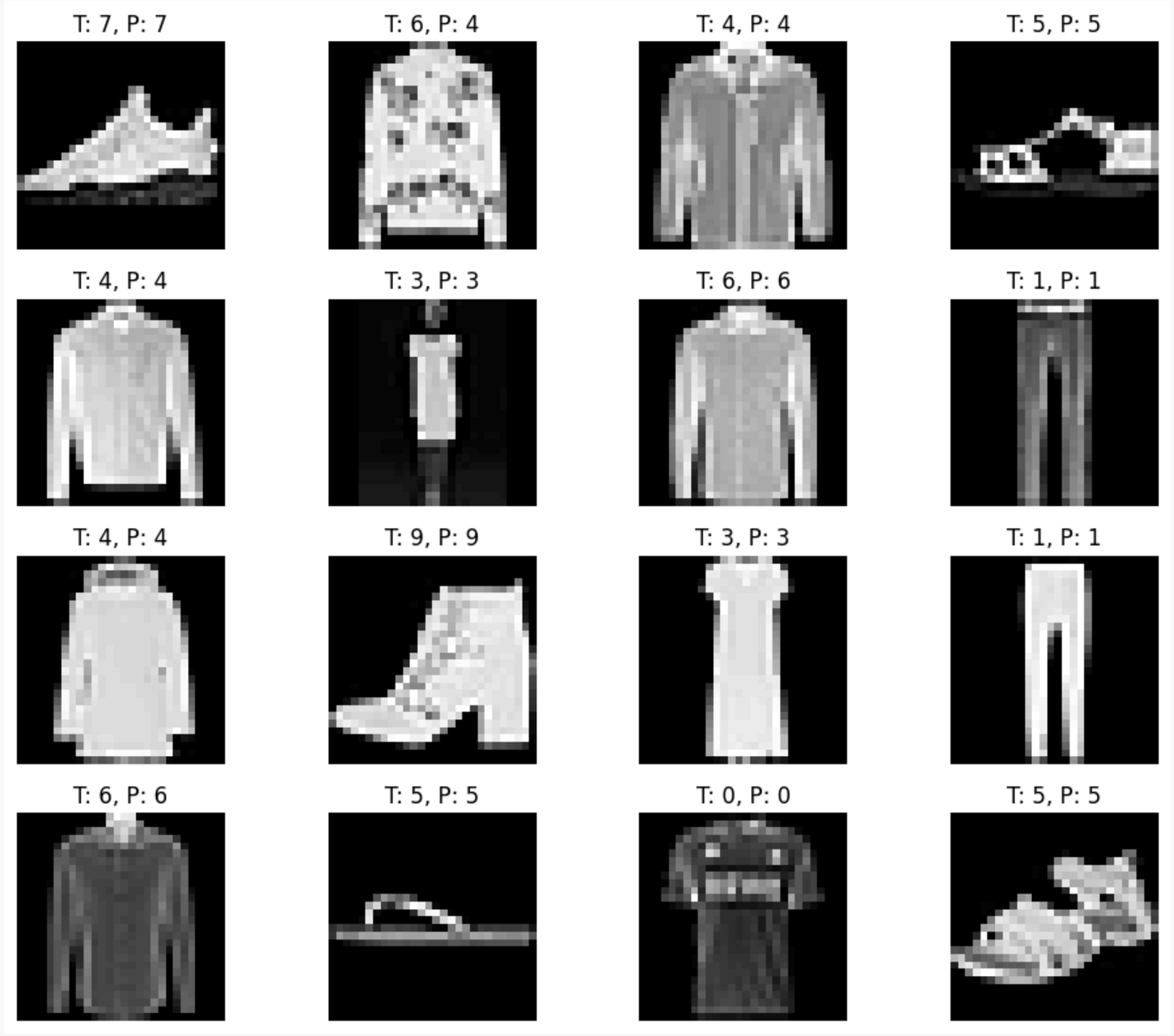
Back to Agenda

```
Epoch 1/30
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.6885 - loss: 0.8948
Epoch 2/30
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8523 - loss: 0.4069
Epoch 3/30
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8790 - loss: 0.3339
Epoch 4/30
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8916 - loss: 0.2971
Epoch 5/30
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.8989 - loss: 0.2775
Epoch 6/30
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.9116 - loss: 0.2444
Epoch 7/30
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9120 - loss: 0.2368
Epoch 8/30
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.9194 - loss: 0.2185
Epoch 9/30
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.9245 - loss: 0.2066
Epoch 10/30
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.9312 - loss: 0.1916
Epoch 11/30
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.9349 - loss: 0.1823
Epoch 12/30
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.9360 - loss: 0.1733
Epoch 13/30
...
Epoch 30/30
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.9815 - loss: 0.0498
313/313 - 1s - 2ms/step - accuracy: 0.9066 - loss: 0.4260
Test accuracy: 0.9065999984741211
```

Back to Agenda

```python
# Get 16 random indices for the test set
#indices = np.random.choice(len(test_images), size=16, replace=False)
#indices = np.where(np.random.rand(len(test_images)) < 0.025)[0]
indices = [1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1034, 1035, 1036, 1037, 1038, 1039]
print(indices)
# Make predictions on the test set
y_pred_logits = model.predict(test_images[indices])
y_pred = np.argmax(y_pred_logits, axis=-1)

# Create a figure with 16 subplots
fig, axs = plt.subplots(nrows=4, ncols=4, figsize=(10, 8))

# For each subplot, plot the corresponding image and write the true and predicted labels on top
for i, ax in enumerate(axs.flatten()):
    if i < len(indices):
        img_index = indices[i]
        ax.imshow(test_images[img_index], cmap='gray')
        ax.set_title(f'T: {test_labels[img_index]}, P: {y_pred[i]}', fontsize=12)
        ax.axis('off')

#for i, ax in zip(indices, axs.flatten()):
#    ax.imshow(test_images[i], cmap='gray')
    #ax.set_title(f'True: {test_labels[i]}, Pred: {y_pred[i]}', fontsize=12)
#    ax.set_title(f'T {test_labels[i]}, P {y_pred[i]}', fontsize=12)
#    ax.axis('off')

plt.tight_layout()
plt.show()
```

Back to Agenda



Back to Agenda

Using our personal ANN-Artificial Neural Network

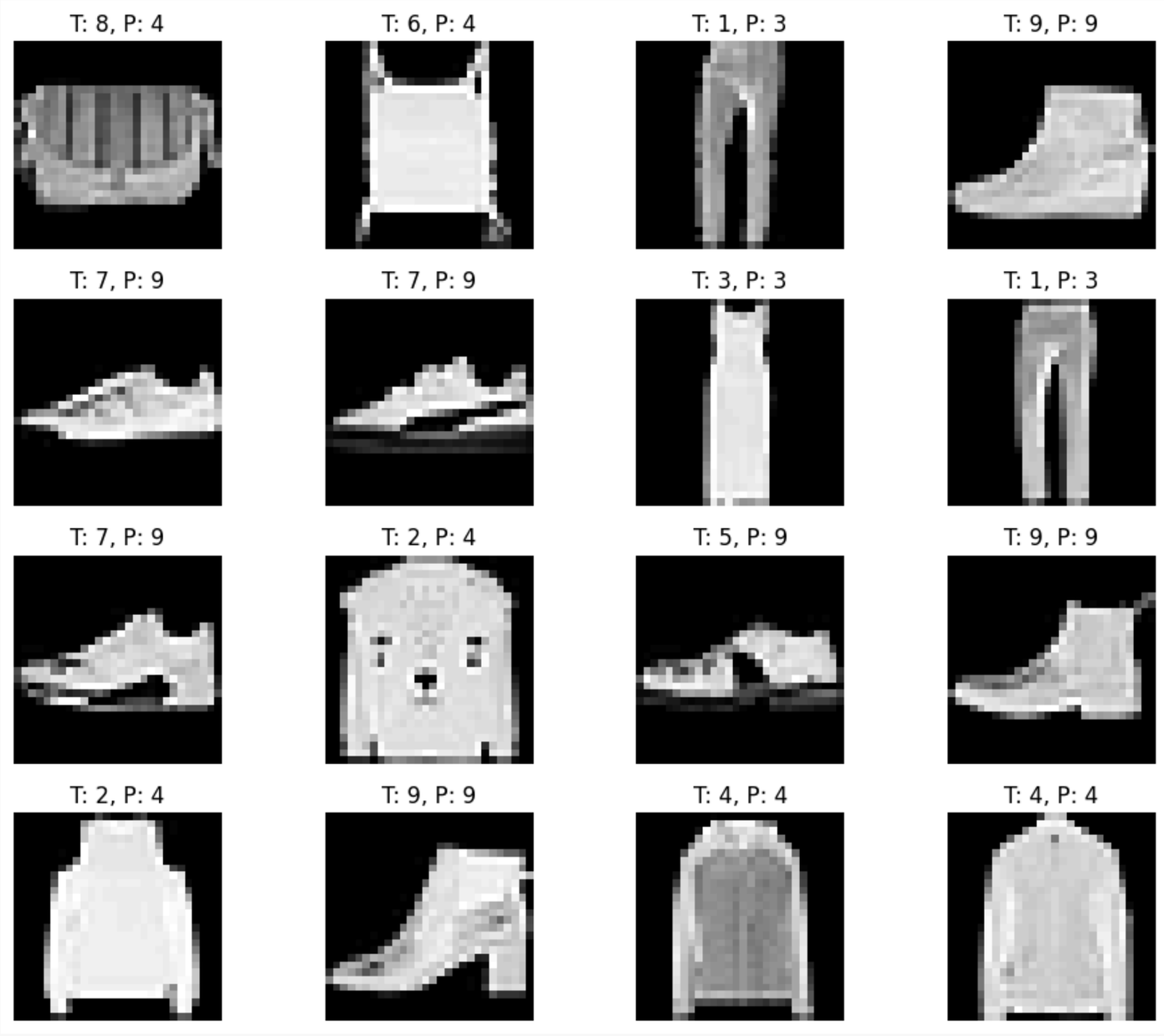
After trying to implement the Fashion-MNIST dataset on our personal code, we added:

- categorical cross-entropy calculation
- one-hot encoding
- batching
- 10 output neurons instead of just 1

Back to Agenda

LR=0.01 - MF=0.9 - structure = [794, 1024, 1024, 512, 10] - batch size = 128 - epochs = 100

```
Epoch 1/100, Accuracy: 40.62%
Epoch 2/100, Accuracy: 62.73%
Epoch 3/100, Accuracy: 66.39%
Epoch 4/100, Accuracy: 68.98%
Epoch 5/100, Accuracy: 70.16%
Epoch 6/100, Accuracy: 71.78%
Epoch 7/100, Accuracy: 73.43%
Epoch 8/100, Accuracy: 73.33%
Epoch 9/100, Accuracy: 74.14%
Epoch 10/100, Accuracy: 75.60%
Epoch 11/100, Accuracy: 76.49%
Epoch 12/100, Accuracy: 76.64%
Epoch 13/100, Accuracy: 77.50%
Epoch 14/100, Accuracy: 77.81%
Epoch 15/100, Accuracy: 78.84%
Epoch 16/100, Accuracy: 78.01%
Epoch 17/100, Accuracy: 78.68%
Epoch 18/100, Accuracy: 79.19%
Epoch 19/100, Accuracy: 79.09%
Epoch 20/100, Accuracy: 79.95%
Epoch 21/100, Accuracy: 80.08%
Epoch 22/100, Accuracy: 80.25%
Epoch 23/100, Accuracy: 80.20%
Epoch 24/100, Accuracy: 80.19%
Epoch 25/100, Accuracy: 80.58%
...
Epoch 50/100, Accuracy: 81.68%
Epoch 51/100, Accuracy: 81.29%
Epoch 52/100, Accuracy: 81.76%
Epoch 53/100, Accuracy: 81.18%
```

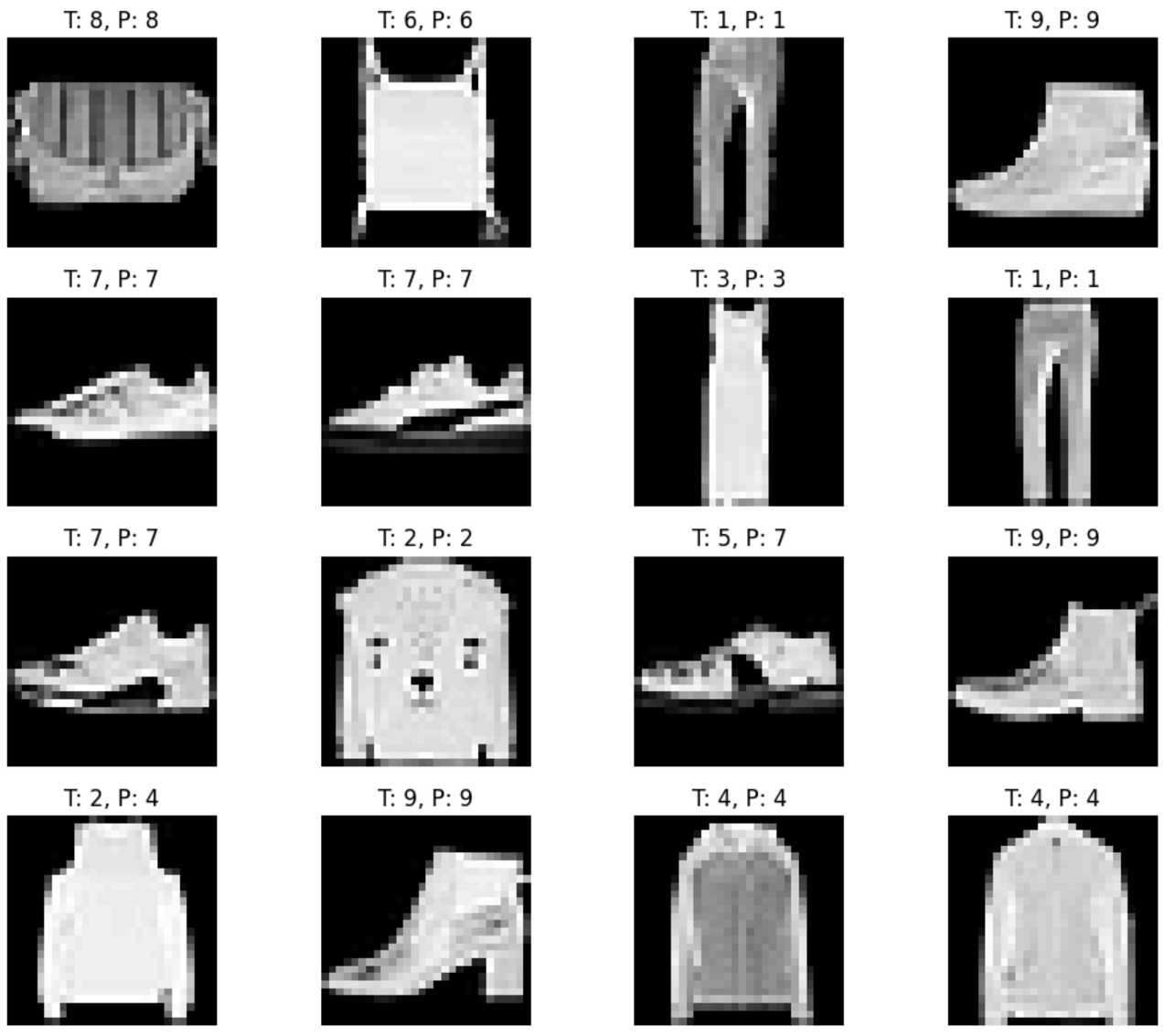


Back to Agenda

LR=0.01 - MF=0.1 - structure = [794, 512, 10] - batch size = 64 - epochs = 10

```
Epoch 1/10, Accuracy: 74.44%, Average Loss: 2.3188
Epoch 2/10, Accuracy: 80.41%, Average Loss: 1.1689
Epoch 3/10, Accuracy: 82.71%, Average Loss: 0.9311
Epoch 4/10, Accuracy: 83.86%, Average Loss: 0.8079
Epoch 5/10, Accuracy: 84.83%, Average Loss: 0.7175
Epoch 6/10, Accuracy: 85.59%, Average Loss: 0.6495
Epoch 7/10, Accuracy: 86.16%, Average Loss: 0.6070
Epoch 8/10, Accuracy: 86.86%, Average Loss: 0.5664
Epoch 9/10, Accuracy: 87.47%, Average Loss: 0.5276
Epoch 10/10, Accuracy: 87.93%, Average Loss: 0.5018
Highest accuracy reached: 87.93%
```

```
Epoch 1/10, Accuracy: 73.78%, Average Loss: 2.4643
Epoch 2/10, Accuracy: 80.17%, Average Loss: 1.1861
Epoch 3/10, Accuracy: 82.35%, Average Loss: 0.9270
Epoch 4/10, Accuracy: 83.59%, Average Loss: 0.8125
Epoch 5/10, Accuracy: 84.72%, Average Loss: 0.7144
Epoch 6/10, Accuracy: 85.45%, Average Loss: 0.6520
Epoch 7/10, Accuracy: 86.34%, Average Loss: 0.5919
Epoch 8/10, Accuracy: 86.93%, Average Loss: 0.5585
Epoch 9/10, Accuracy: 87.50%, Average Loss: 0.5211
Epoch 10/10, Accuracy: 88.03%, Average Loss: 0.4845
Highest accuracy reached: 88.03%
```

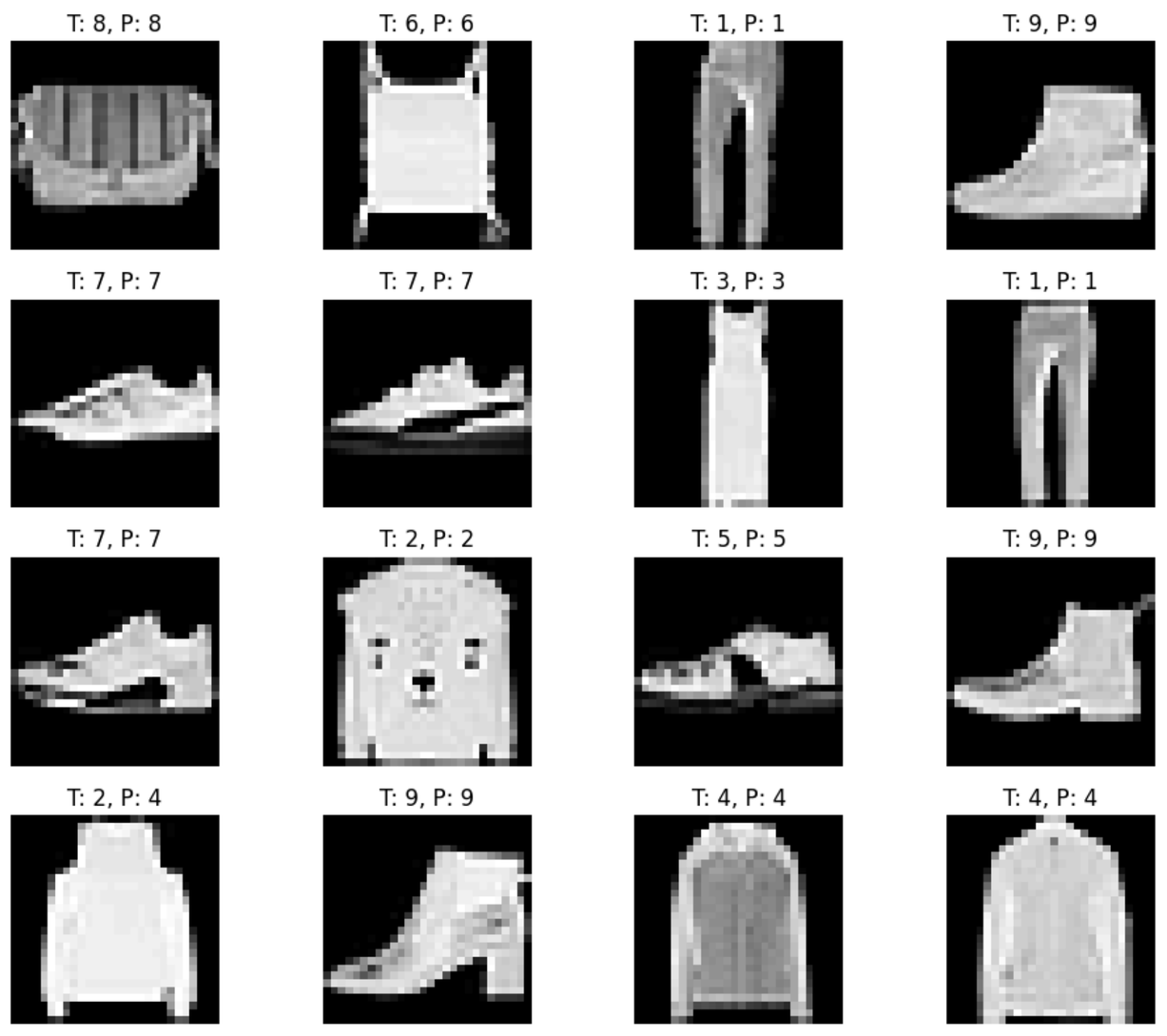


Back to Agenda

LR=0.01 - MF=0.9 - structure = [794, 512, 10] - batch size = 32 - epochs = 100

Training accuracy of 99.61%

```
Epoch 1/100, Accuracy: 76.61%, Average Loss: 1.3812
Epoch 2/100, Accuracy: 81.96%, Average Loss: 0.8205
Epoch 3/100, Accuracy: 83.87%, Average Loss: 0.6853
Epoch 4/100, Accuracy: 85.11%, Average Loss: 0.6030
Epoch 5/100, Accuracy: 85.93%, Average Loss: 0.5503
Epoch 6/100, Accuracy: 86.59%, Average Loss: 0.5107
Epoch 7/100, Accuracy: 87.30%, Average Loss: 0.4765
Epoch 8/100, Accuracy: 87.83%, Average Loss: 0.4494
Epoch 9/100, Accuracy: 88.38%, Average Loss: 0.4273
Epoch 10/100, Accuracy: 88.90%, Average Loss: 0.4064
Epoch 11/100, Accuracy: 89.33%, Average Loss: 0.3877
Epoch 12/100, Accuracy: 89.66%, Average Loss: 0.3704
Epoch 13/100, Accuracy: 90.09%, Average Loss: 0.3531
Epoch 14/100, Accuracy: 90.46%, Average Loss: 0.3386
Epoch 15/100, Accuracy: 90.77%, Average Loss: 0.3254
Epoch 16/100, Accuracy: 91.11%, Average Loss: 0.3131
Epoch 17/100, Accuracy: 91.39%, Average Loss: 0.3017
Epoch 18/100, Accuracy: 91.69%, Average Loss: 0.2913
Epoch 19/100, Accuracy: 91.98%, Average Loss: 0.2812
Epoch 20/100, Accuracy: 92.27%, Average Loss: 0.2717
Epoch 21/100, Accuracy: 92.54%, Average Loss: 0.2627
Epoch 22/100, Accuracy: 92.80%, Average Loss: 0.2542
Epoch 23/100, Accuracy: 93.07%, Average Loss: 0.2460
Epoch 24/100, Accuracy: 93.29%, Average Loss: 0.2380
Epoch 25/100, Accuracy: 93.54%, Average Loss: 0.2304
...
Epoch 98/100, Accuracy: 99.58%, Average Loss: 0.0334
Epoch 99/100, Accuracy: 99.60%, Average Loss: 0.0326
Epoch 100/100, Accuracy: 99.61%, Average Loss: 0.0320
Highest accuracy reached: 99.61%
```



Back to Agenda

Our Code

Back to Agenda

Thank you all!