

```
In [1]: #import libraries
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import datasets
from tensorflow.keras import layers
```

```
In [2]: #####
#
# TO DO:
#     load fashion mnist dataset's training/testing samples and targets
#     retype input samples as float 32
#     check the shape of the training/testing samples and targets

(x_train, y_train), (x_test, y_test) = datasets.fashion_mnist.load_data()

num_train_samples, nx, ny = x_train.shape
num_test_samples = len(x_test)

x_train = x_train.astype("float32")
x_test = x_test.astype("float32")

print(f"TRAINING input sample shape = {x_train.shape}")
print(f"TRAINING target shape = {y_train.shape}")

print(f"\nTESTING input sample shape = {x_test.shape}")
print(f"TESTING target shape = {y_test.shape}")
```

TRAINING input sample shape = (60000, 28, 28)

TRAINING target shape = (60000,)

TESTING input sample shape = (10000, 28, 28)

TESTING target shape = (10000,)

```
In [3]: #####
#
# TO DO:
#     Find an image for each type of garment and display that image
#     with the title of that garment over the top.
#     Arrange your images in a 2 (rows) by 5 (columns) grid.

fig, axs = plt.subplots(2,5)
dictlist = ('T-shirt', 'Trouser', 'Pullover', 'Dress', 'Coat',
            'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Angle Boot')

for label in range(10):
    indx = np.argwhere(y_test==label)
    nentry = indx[0]
    irow = np.floor(label/5).astype('uint8')
    icol = np.floor(label-5*irow).astype('uint8')
    image = x_test[nentry[0]]
```

```

axs[irow,icol].axis("off")
axs[irow,icol].matshow(image,cmap="gray")
axs[irow,icol].set_title(dictlist[label])

```

T-shirt



Trouser



Pullover



Dress



Coat



Sandal



Shirt



Sneaker



Bag



Angle Boot



```

In [4]: #####
#
# TO DO:
#   Instantiate TensorFlow sequential model with two dense layers
#   The first dense layer should have 128 nodes, using ReLu activation
#   Second layer has 10 nodes using softmax activation
#   compile the model with an adam optimizer, sparse categorical
#   crossentropy loss, with "accuracy" as the performanc metric
#
#   print model summary

inputs = keras.Input(shape = (28*28,))
x = layers.Dense(128, activation="relu")(inputs)
outputs = layers.Dense(10,activation="softmax")(x)
model = keras.Model(inputs=inputs,outputs=outputs)
model.summary()

model.compile(optimizer="adam",
              loss = "sparse_categorical_crossentropy",
              metrics = ["accuracy"])

```

**Model: "functional"**

Layer (type)	Output Shape	Par
input_layer (InputLayer)	(None, 784)	
dense (Dense)	(None, 128)	100
dense_1 (Dense)	(None, 10)	1

**Total params:** 101,770 (397.54 KB)

**Trainable params:** 101,770 (397.54 KB)

**Non-trainable params:** 0 (0.00 B)

```
In [5]: #####
#
# TO DO:
#     reshape the training and test samples to (60000,28*28)
#     rescale the samples to [0,1]
#     split the training data into a p-training and validation
#         using a 2/3 split
#     set a callback to save the "best-model" with lowest
#         validation loss
#     train the model on p-training data for 30 epochs assuming
#         batch size of 512 and returning the history object


x_train_n = np.array(x_train).astype("float32")/255
x_train_n = x_train_n.reshape(60000,28*28)
N,_ = x_train_n.shape
train_split = 2/3
Nptrain = np.ceil(N*train_split).astype("uint32")


x_ptrain = x_train_n[:Nptrain]
y_ptrain = y_train[:Nptrain]
x_val     = x_train_n[Nptrain:]
y_val     = y_train[Nptrain:]


x_test_n = np.array(x_test).astype("float32")/255
x_test_n = x_test_n.reshape(10000,28*28)


callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="model_sequential.keras",
        save_best_only = True,
        monitor = "val_loss"
    )
]


history = model.fit(x_ptrain, y_ptrain,
                    epochs = 30,
                    batch_size=512,
                    validation_data = (x_val,y_val),
                    callbacks = callbacks)
```


Epoch 1/30  
79/79  1s 4ms/step - accuracy: 0.6135 - loss: 1.1632 - val\_accuracy: 0.8149 - val\_loss: 0.5481


Epoch 2/30  
79/79  0s 3ms/step - accuracy: 0.8232 - loss: 0.5212 - val\_accuracy: 0.8404 - val\_loss: 0.4707


Epoch 3/30  
79/79  0s 3ms/step - accuracy: 0.8432 - loss: 0.4599 - val\_accuracy: 0.8433 - val\_loss: 0.4472


Epoch 4/30  
79/79  0s 3ms/step - accuracy: 0.8536 - loss: 0.4242 - val\_accuracy: 0.8495 - val\_loss: 0.4268


Epoch 5/30  
79/79  0s 3ms/step - accuracy: 0.8573 - loss: 0.4063 - val\_accuracy: 0.8529 - val\_loss: 0.4182


Epoch 6/30  
79/79  0s 4ms/step - accuracy: 0.8641 - loss: 0.3890 - val\_accuracy: 0.8478 - val\_loss: 0.4260


Epoch 7/30  
79/79  0s 3ms/step - accuracy: 0.8720 - loss: 0.3690 - val\_accuracy: 0.8557 - val\_loss: 0.4118


Epoch 8/30  
79/79  0s 4ms/step - accuracy: 0.8725 - loss: 0.3667 - val\_accuracy: 0.8683 - val\_loss: 0.3800


Epoch 9/30  
79/79  0s 3ms/step - accuracy: 0.8788 - loss: 0.3455 - val\_accuracy: 0.8702 - val\_loss: 0.3689


Epoch 10/30  
79/79  0s 4ms/step - accuracy: 0.8791 - loss: 0.3350 - val\_accuracy: 0.8669 - val\_loss: 0.3754


Epoch 11/30  
79/79  0s 3ms/step - accuracy: 0.8826 - loss: 0.3282 - val\_accuracy: 0.8733 - val\_loss: 0.3647


Epoch 12/30  
79/79  0s 3ms/step - accuracy: 0.8862 - loss: 0.3234 - val\_accuracy: 0.8738 - val\_loss: 0.3574


Epoch 13/30  
79/79  0s 4ms/step - accuracy: 0.8909 - loss: 0.3054 - val\_accuracy: 0.8742 - val\_loss: 0.3539


Epoch 14/30  
79/79  0s 4ms/step - accuracy: 0.8904 - loss: 0.2990 - val\_accuracy: 0.8756 - val\_loss: 0.3517

Epoch 15/30  
79/79  0s 4ms/step - accuracy: 0.8939 - loss: 0.2957 - val\_accuracy: 0.8794 - val\_loss: 0.3416












Epoch 16/30  
79/79  0s 3ms/step - accuracy: 0.8950 - loss: 0.2904 - val\_accuracy: 0.8783 - val\_loss: 0.3470

Epoch 17/30  
79/79  0s 3ms/step - accuracy: 0.8981 - loss: 0.2813 - val\_accuracy: 0.8827 - val\_loss: 0.3356

Epoch 18/30  
79/79  0s 3ms/step - accuracy: 0.9001 - loss: 0.2774 - val\_accuracy: 0.8807 - val\_loss: 0.3331

Epoch 19/30  
79/79  0s 3ms/step - accuracy: 0.9025 - loss: 0.2731 - val\_accuracy: 0.8807 - val\_loss: 0.3331

```

al_accuracy: 0.8719 - val_loss: 0.3530
Epoch 20/30
79/79  0s 3ms/step - accuracy: 0.9029 - loss: 0.2666 - v
al_accuracy: 0.8836 - val_loss: 0.3296
Epoch 21/30
79/79  0s 3ms/step - accuracy: 0.9066 - loss: 0.2584 - v
al_accuracy: 0.8803 - val_loss: 0.3430
Epoch 22/30
79/79  0s 3ms/step - accuracy: 0.9053 - loss: 0.2593 - v
al_accuracy: 0.8862 - val_loss: 0.3211
Epoch 23/30
79/79  0s 3ms/step - accuracy: 0.9113 - loss: 0.2483 - v
al_accuracy: 0.8867 - val_loss: 0.3194
Epoch 24/30
79/79  0s 3ms/step - accuracy: 0.9116 - loss: 0.2450 - v
al_accuracy: 0.8784 - val_loss: 0.3404
Epoch 25/30
79/79  0s 3ms/step - accuracy: 0.9078 - loss: 0.2494 - v
al_accuracy: 0.8802 - val_loss: 0.3351
Epoch 26/30
79/79  0s 3ms/step - accuracy: 0.9151 - loss: 0.2398 - v
al_accuracy: 0.8851 - val_loss: 0.3245
Epoch 27/30
79/79  0s 3ms/step - accuracy: 0.9179 - loss: 0.2309 - v
al_accuracy: 0.8858 - val_loss: 0.3181
Epoch 28/30
79/79  0s 3ms/step - accuracy: 0.9187 - loss: 0.2284 - v
al_accuracy: 0.8885 - val_loss: 0.3140
Epoch 29/30
79/79  0s 3ms/step - accuracy: 0.9183 - loss: 0.2298 - v
al_accuracy: 0.8842 - val_loss: 0.3293
Epoch 30/30
79/79  0s 3ms/step - accuracy: 0.9213 - loss: 0.2229 - v
al_accuracy: 0.8853 - val_loss: 0.3262

```

```

In [6]: #####
#
# TO DO:
#     create a function "training_curves" that takes "history"
#     and "best_model" as inputs
#     your function should plots ptraining/validation loss vs epoch
#           ptraining/validation accuracy vs epoch
#     then evaluate the "best model" accuracy on the test dataset

#plot training curves
def plot_training_curves(history):
    history_dict = history.history
    loss_values = history_dict["loss"]
    val_loss_values = history_dict["val_loss"]
    acc_values = history_dict["accuracy"]
    val_acc_values = history_dict["val_accuracy"]
    epochs = range(1, len(loss_values)+1)

    test_loss, test_acc

```

```

figure, axis = plt.subplots(1,2,figsize=(10,4))
axis[0].plot(epochs, loss_values, "bo", label="training loss")
axis[0].plot(epochs, val_loss_values, "b", label="Validation loss")
axis[0].set_title("pTraining and Val Losses")
axis[0].set_xlabel("Epochs")
axis[0].set_ylabel("Loss")
axis[0].legend()

axis[1].plot(epochs, acc_values, "bo", label="Training acc")
axis[1].plot(epochs, val_acc_values, "b", label="Validation acc")
axis[1].set_title("ptraining and val accuracy")
axis[1].set_xlabel("Epochs")
axis[1].set_ylabel("Accuracy")
axis[1].legend()

best_model = keras.models.load_model("model_sequential.keras")
test_loss, test_acc = best_model.evaluate(x_test_n,y_test, verbose=1)
print(f"Test Accuracy = {test_acc*100:.2f}%")
plot_training_curves(history)

```

313/313 ————— 0s 515us/step – accuracy: 0.8813 – loss: 0.3392  
 Test Accuracy = 88.07%

