problem2

November 29, 2023

[ ]:
```python
# Import packages
# DL Packages
import tensorflow as tf
import keras

# Others
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
import sympy as sym
import seaborn as sns

from sklearn.metrics import confusion_matrix
```

```
2023-11-29 13:42:51.561847: I tensorflow/core/util/port.cc:111] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2023-11-29 13:42:51.584741: E
tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:9342] Unable to
register cuDNN factory: Attempting to register factory for plugin cuDNN when one
has already been registered
2023-11-29 13:42:51.584763: E
tensorflow/compiler/xla/stream_executor/cuda/cuda_fft.cc:609] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2023-11-29 13:42:51.584774: E
tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:1518] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2023-11-29 13:42:51.588668: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
```

1 Examine the Data:

Begin by again looking at the shapes

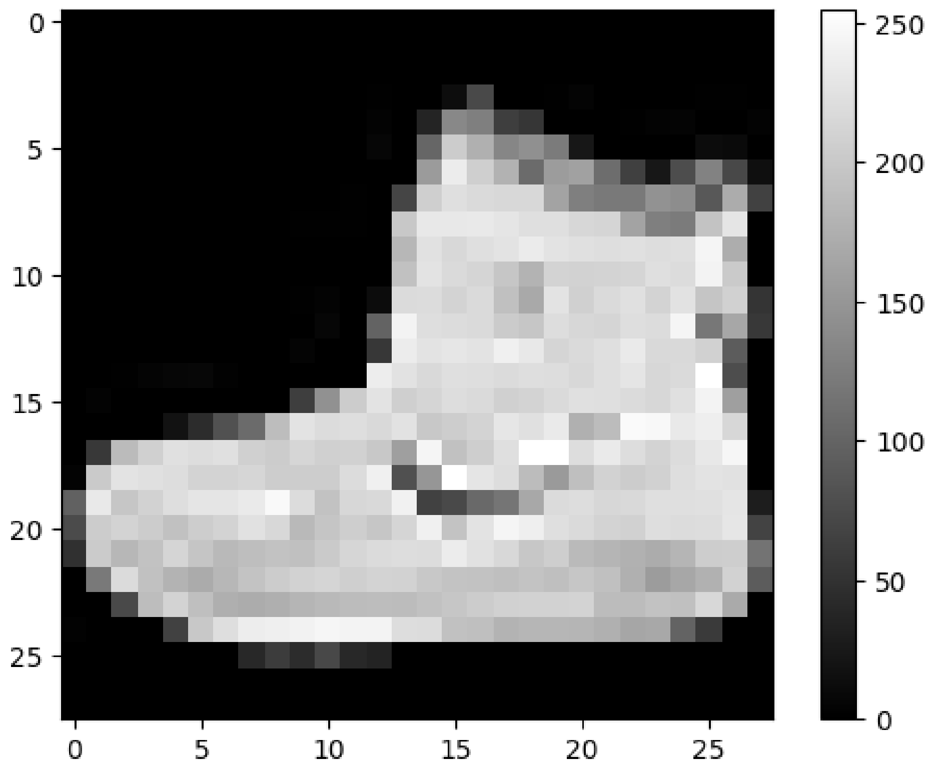
[ ]:
```
(Xtrain, Ytrain), (Xtest, Ytest) = tf.keras.datasets.fashion_mnist.load_data()
```

[ ]:
```
print("Xtrain shape: ", Xtrain.shape)
print("Xtrain min, max: ", Xtrain.min(), Xtrain.max())
print("-----------------------")
print("Ytrain sthape: ", Ytrain.shape)
print("Ytrain classes: ", np.unique(Ytrain))
print("-----------------------")
print("Xtest.shape: ", Xtest.shape)
print("Xtest min, max: ", Xtrain.min(), Xtrain.max())
print("-----------------------")
print("Ytest shape: ", Ytest.shape)
print("Ytest classes: ", np.unique(Ytrain))
```

```
Xtrain shape:  (60000, 28, 28)
Xtrain min, max:  0 255
-----------------------
Ytrain sthape:  (60000,)
Ytrain classes:  [0 1 2 3 4 5 6 7 8 9]
-----------------------
Xtest.shape:  (10000, 28, 28)
Xtest min, max:  0 255
-----------------------
Ytest shape:  (10000,)
Ytest classes:  [0 1 2 3 4 5 6 7 8 9]
```

Same exact shapes and sizes and number of classes as the first problem. This means we're expecting 28x28 grayscale images, but suppodely this one is "fashion" so let's see why.

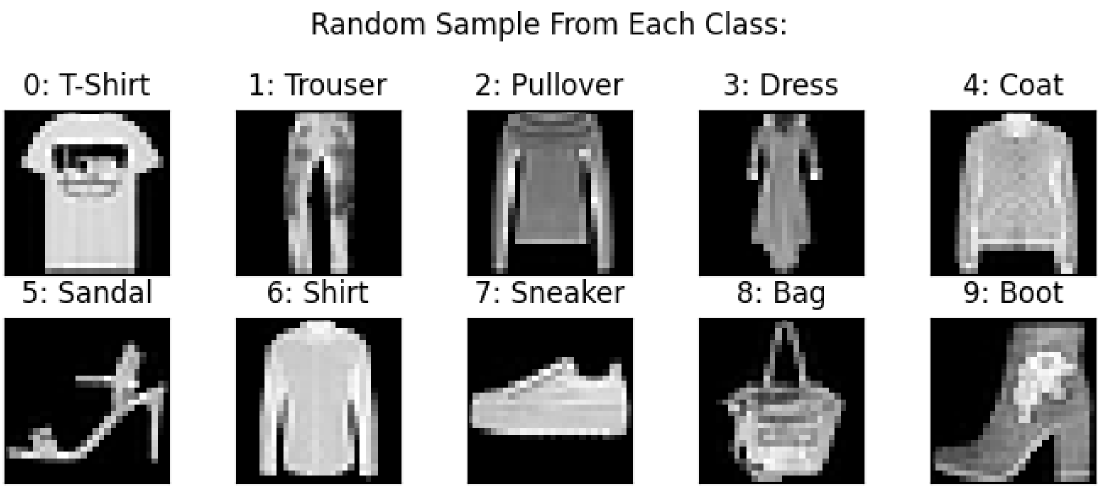
[ ]:
```
plt.imshow(Xtrain[0], cmap="gray")
plt.colorbar();
```



It's a boot! Let's look at the all the classes like last time. I got the mappings from labels to classes from here

[ ]:
```
ncols = 5
nrows = 2
label_vals = np.unique(Ytrain)
# By label val index
mnist_labels = ["T-Shirt", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt", "Sneaker", "Bag", "Boot"]
f, ax = plt.subplots(nrows=nrows, ncols=ncols)
f.set_size_inches(7,3)
plt.suptitle("Random Sample From Each Class:")
for i in range(nrows):
    for j in range(ncols):
        n = label_vals[i*ncols + j]
        is_n = np.nonzero(Ytrain==n)[0]
        random_i = np.random.choice(is_n)
        ax[i,j].imshow(Xtrain[random_i], origin="upper", cmap="gray")
        ax[i,j].set_aspect(1)
        ax[i,j].get_xaxis().set_visible(False)
        ax[i,j].get_yaxis().set_visible(False)
        ax[i,j].set_title(f"{n}: {mnist_labels[n]}")
```

```
plt.tight_layout()
```



2 Pre-Process Data:

First we will normalize the data to be between 0-1. NO flattening this time, but we'll add an extra dimension to make it (28,28,1). We will use one hot encoding to represent the data labels.

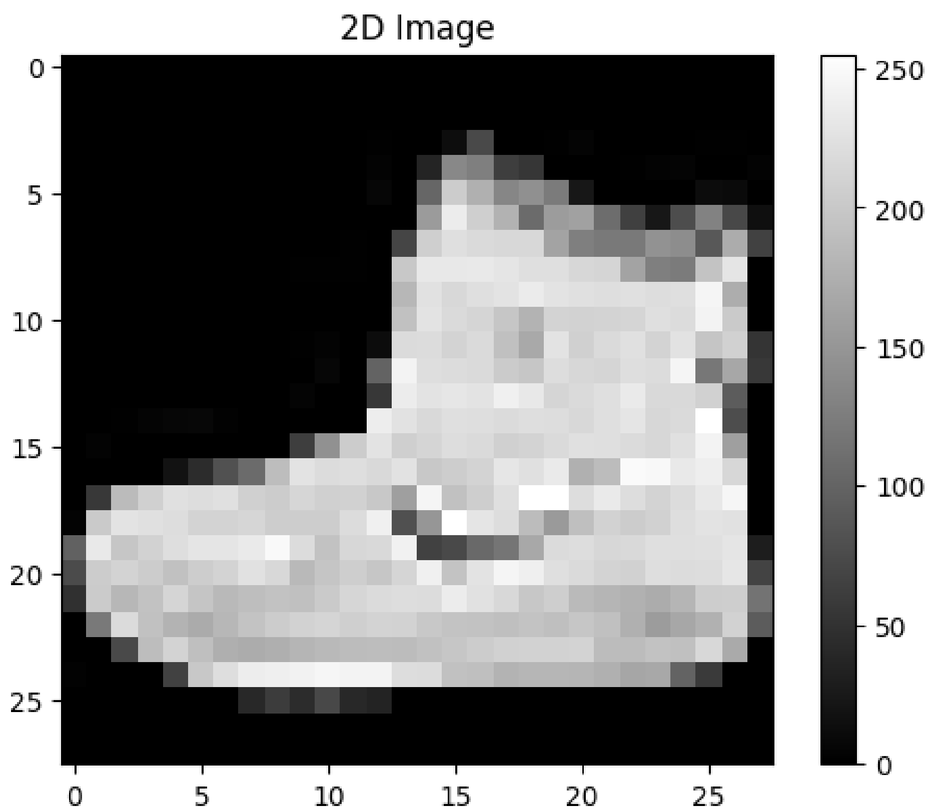
[ ]:
```
Xtrain_norm = np.expand_dims(Xtrain.astype(float)/np.max(Xtrain),-1)
Xtest_norm = np.expand_dims(Xtest.astype(float)/np.max(Xtest),-1)
def OHE(labels):

    Y = np.zeros((labels.size, 1), dtype=int)
    unique_vals = np.unique(labels)
    label_map = {}
    for i, val in enumerate(unique_vals):
        label_map[val] = i
    count = 0
    for i, label in enumerate(labels):
        # Assign label
        Y[i] = label_map[label]

    Y_OHE = np.zeros((Y.shape[0], len(label_map)), dtype=int)
    for i in range(Y.shape[0]):
        Y_OHE[i, Y[i]] = 1
    Y = Y_OHE

    return Y_OHE

Ytrain_OHE = OHE(Ytrain)
```

```
Ytest_OHE = OHE(Ytest)
```

To verify everything worked, look at the first sample:

[ ]:
```
print("Ytrain[0]:", f"{Ytrain[0]},", mnist_labels[Ytrain[0]], " Ytrain_OHE[0]:
 ↪", Ytrain_OHE[0])
plt.imshow(Xtrain[0], cmap="gray")
plt.title("2D Image")
plt.colorbar();
```

```
Ytrain[0]: 9, Boot  Ytrain_OHE[0]: [0 0 0 0 0 0 0 0 0 1]
```



Looks good again!

3 Make/Train a Network:

[ ]:
```
input_size = (28, 28, 1)
output_size = 10 # one hot encoded label vals
model = keras.models.Sequential([
    keras.Input(shape=input_size),
```

```python
    keras.layers.Conv2D(filters=16, kernel_size=(3,3), strides=(1,1),␣
↪padding="same", activation="relu"),
    keras.layers.MaxPool2D(pool_size=(2,2)),
    keras.layers.Conv2D(filters=8, kernel_size=(3,3), strides=(1,1),␣
↪padding="same", activation="relu"),
    keras.layers.MaxPool2D(pool_size=(2,2)),
    keras.layers.Conv2D(filters=4, kernel_size=(3,3), strides=(1,1),␣
↪padding="same", activation="relu"),
    keras.layers.Flatten(),
    keras.layers.Dropout(0.1),
    keras.layers.Dense(output_size, activation="softmax")
], name="mnist_dense")
model.build(input_size)
model.compile(optimizer="adam", loss="categorical_crossentropy",
              metrics=[keras.metrics.CategoricalAccuracy()])
model.summary()
```

```
Model: "mnist_dense"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 16)        160

 max_pooling2d (MaxPooling2  (None, 14, 14, 16)        0
 D)

 conv2d_1 (Conv2D)           (None, 14, 14, 8)         1160

 max_pooling2d_1 (MaxPoolin  (None, 7, 7, 8)           0
 g2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 4)           292

 flatten (Flatten)           (None, 196)               0

 dropout (Dropout)           (None, 196)               0

 dense (Dense)               (None, 10)                1970

=================================================================
Total params: 3582 (13.99 KB)
Trainable params: 3582 (13.99 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________

2023-11-29 13:42:53.796485: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:894]
successful NUMA node read from SysFS had negative value (-1), but there must be
```

```
at least one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-pci#L344-L355
2023-11-29 13:42:53.799381: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:894]
successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-pci#L344-L355
2023-11-29 13:42:53.799469: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:894]
successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-pci#L344-L355
2023-11-29 13:42:53.800431: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:894]
successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-pci#L344-L355
2023-11-29 13:42:53.800526: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:894]
successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-pci#L344-L355
2023-11-29 13:42:53.800585: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:894]
successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-pci#L344-L355
2023-11-29 13:42:53.840550: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:894]
successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-pci#L344-L355
2023-11-29 13:42:53.840643: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:894]
successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-pci#L344-L355
2023-11-29 13:42:53.840710: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:894]
successful NUMA node read from SysFS had negative value (-1), but there must be
```

```
at least one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2023-11-29 13:42:53.840762: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1886] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 917 MB memory:  -> device: 0,
name: NVIDIA RTX A4500, pci bus id: 0000:01:00.0, compute capability: 8.6
```

[ ]:
```
history = model.fit(Xtrain_norm, Ytrain_OHE, batch_size=1000, epochs=100,␣
↪validation_data=(Xtest_norm, Ytest_OHE))
```

```
Epoch 1/100
```

```
2023-11-29 13:42:54.499413: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:442] Loaded cuDNN
version 8700
2023-11-29 13:42:54.675356: I tensorflow/tsl/platform/default/subprocess.cc:304]
Start cannot spawn child process: No such file or directory
2023-11-29 13:42:54.944546: I tensorflow/tsl/platform/default/subprocess.cc:304]
Start cannot spawn child process: No such file or directory
2023-11-29 13:42:55.214051: I tensorflow/compiler/xla/service/service.cc:168]
XLA service 0x7fb5415f5410 initialized for platform CUDA (this does not
guarantee that XLA will be used). Devices:
2023-11-29 13:42:55.214084: I tensorflow/compiler/xla/service/service.cc:176]
StreamExecutor device (0): NVIDIA RTX A4500, Compute Capability 8.6
2023-11-29 13:42:55.217002: I
tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR
crash reproducer, set env var `MLIR_CRASH_REPRODUCER_DIRECTORY` to enable.
2023-11-29 13:42:55.260967: I ./tensorflow/compiler/jit/device_compiler.h:186]
Compiled cluster using XLA!  This line is logged at most once for the lifetime
of the process.
```

```
60/60 [==============================] - 3s 6ms/step - loss: 1.8019 -
categorical_accuracy: 0.3637 - val_loss: 0.9490 - val_categorical_accuracy:
0.6536
Epoch 2/100
60/60 [==============================] - 0s 4ms/step - loss: 0.8715 -
categorical_accuracy: 0.6796 - val_loss: 0.7035 - val_categorical_accuracy:
0.7526
Epoch 3/100
60/60 [==============================] - 0s 4ms/step - loss: 0.6941 -
categorical_accuracy: 0.7489 - val_loss: 0.6097 - val_categorical_accuracy:
0.7797
Epoch 4/100
60/60 [==============================] - 0s 4ms/step - loss: 0.6075 -
categorical_accuracy: 0.7814 - val_loss: 0.5505 - val_categorical_accuracy:
0.7995
Epoch 5/100
60/60 [==============================] - 0s 4ms/step - loss: 0.5483 -
```

```
categorical_accuracy: 0.8020 - val_loss: 0.5077 - val_categorical_accuracy:
0.8143
Epoch 6/100
60/60 [==============================] - 0s 4ms/step - loss: 0.5094 -
categorical_accuracy: 0.8143 - val_loss: 0.4769 - val_categorical_accuracy:
0.8262
Epoch 7/100
60/60 [==============================] - 0s 4ms/step - loss: 0.4833 -
categorical_accuracy: 0.8240 - val_loss: 0.4562 - val_categorical_accuracy:
0.8341
Epoch 8/100
60/60 [==============================] - 0s 4ms/step - loss: 0.4622 -
categorical_accuracy: 0.8329 - val_loss: 0.4390 - val_categorical_accuracy:
0.8429
Epoch 9/100
60/60 [==============================] - 0s 4ms/step - loss: 0.4468 -
categorical_accuracy: 0.8377 - val_loss: 0.4255 - val_categorical_accuracy:
0.8466
Epoch 10/100
60/60 [==============================] - 0s 4ms/step - loss: 0.4333 -
categorical_accuracy: 0.8434 - val_loss: 0.4148 - val_categorical_accuracy:
0.8534
Epoch 11/100
60/60 [==============================] - 0s 4ms/step - loss: 0.4225 -
categorical_accuracy: 0.8476 - val_loss: 0.4074 - val_categorical_accuracy:
0.8559
Epoch 12/100
60/60 [==============================] - 0s 4ms/step - loss: 0.4143 -
categorical_accuracy: 0.8494 - val_loss: 0.3982 - val_categorical_accuracy:
0.8584
Epoch 13/100
60/60 [==============================] - 0s 4ms/step - loss: 0.4056 -
categorical_accuracy: 0.8527 - val_loss: 0.3912 - val_categorical_accuracy:
0.8618
Epoch 14/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3984 -
categorical_accuracy: 0.8559 - val_loss: 0.3855 - val_categorical_accuracy:
0.8653
Epoch 15/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3929 -
categorical_accuracy: 0.8578 - val_loss: 0.3805 - val_categorical_accuracy:
0.8635
Epoch 16/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3870 -
categorical_accuracy: 0.8602 - val_loss: 0.3745 - val_categorical_accuracy:
0.8656
Epoch 17/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3809 -
```

```
categorical_accuracy: 0.8631 - val_loss: 0.3691 - val_categorical_accuracy:
0.8694
Epoch 18/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3753 -
categorical_accuracy: 0.8650 - val_loss: 0.3642 - val_categorical_accuracy:
0.8693
Epoch 19/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3702 -
categorical_accuracy: 0.8663 - val_loss: 0.3616 - val_categorical_accuracy:
0.8724
Epoch 20/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3655 -
categorical_accuracy: 0.8671 - val_loss: 0.3580 - val_categorical_accuracy:
0.8710
Epoch 21/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3606 -
categorical_accuracy: 0.8686 - val_loss: 0.3557 - val_categorical_accuracy:
0.8728
Epoch 22/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3579 -
categorical_accuracy: 0.8696 - val_loss: 0.3525 - val_categorical_accuracy:
0.8750
Epoch 23/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3557 -
categorical_accuracy: 0.8715 - val_loss: 0.3477 - val_categorical_accuracy:
0.8777
Epoch 24/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3514 -
categorical_accuracy: 0.8717 - val_loss: 0.3473 - val_categorical_accuracy:
0.8759
Epoch 25/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3477 -
categorical_accuracy: 0.8736 - val_loss: 0.3432 - val_categorical_accuracy:
0.8789
Epoch 26/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3458 -
categorical_accuracy: 0.8742 - val_loss: 0.3419 - val_categorical_accuracy:
0.8767
Epoch 27/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3413 -
categorical_accuracy: 0.8763 - val_loss: 0.3389 - val_categorical_accuracy:
0.8785
Epoch 28/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3392 -
categorical_accuracy: 0.8765 - val_loss: 0.3368 - val_categorical_accuracy:
0.8817
Epoch 29/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3363 -
```

```
categorical_accuracy: 0.8777 - val_loss: 0.3371 - val_categorical_accuracy:
0.8779
Epoch 30/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3363 -
categorical_accuracy: 0.8771 - val_loss: 0.3346 - val_categorical_accuracy:
0.8804
Epoch 31/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3349 -
categorical_accuracy: 0.8773 - val_loss: 0.3333 - val_categorical_accuracy:
0.8818
Epoch 32/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3314 -
categorical_accuracy: 0.8800 - val_loss: 0.3292 - val_categorical_accuracy:
0.8823
Epoch 33/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3306 -
categorical_accuracy: 0.8804 - val_loss: 0.3295 - val_categorical_accuracy:
0.8828
Epoch 34/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3273 -
categorical_accuracy: 0.8817 - val_loss: 0.3267 - val_categorical_accuracy:
0.8845
Epoch 35/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3272 -
categorical_accuracy: 0.8816 - val_loss: 0.3252 - val_categorical_accuracy:
0.8836
Epoch 36/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3247 -
categorical_accuracy: 0.8825 - val_loss: 0.3277 - val_categorical_accuracy:
0.8821
Epoch 37/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3235 -
categorical_accuracy: 0.8828 - val_loss: 0.3276 - val_categorical_accuracy:
0.8820
Epoch 38/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3225 -
categorical_accuracy: 0.8832 - val_loss: 0.3234 - val_categorical_accuracy:
0.8851
Epoch 39/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3202 -
categorical_accuracy: 0.8839 - val_loss: 0.3220 - val_categorical_accuracy:
0.8854
Epoch 40/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3187 -
categorical_accuracy: 0.8837 - val_loss: 0.3188 - val_categorical_accuracy:
0.8864
Epoch 41/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3156 -
```

```
categorical_accuracy: 0.8856 - val_loss: 0.3184 - val_categorical_accuracy:
0.8854
Epoch 42/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3143 -
categorical_accuracy: 0.8860 - val_loss: 0.3184 - val_categorical_accuracy:
0.8857
Epoch 43/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3151 -
categorical_accuracy: 0.8868 - val_loss: 0.3229 - val_categorical_accuracy:
0.8828
Epoch 44/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3130 -
categorical_accuracy: 0.8869 - val_loss: 0.3179 - val_categorical_accuracy:
0.8874
Epoch 45/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3104 -
categorical_accuracy: 0.8876 - val_loss: 0.3154 - val_categorical_accuracy:
0.8849
Epoch 46/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3116 -
categorical_accuracy: 0.8870 - val_loss: 0.3147 - val_categorical_accuracy:
0.8863
Epoch 47/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3101 -
categorical_accuracy: 0.8883 - val_loss: 0.3141 - val_categorical_accuracy:
0.8884
Epoch 48/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3105 -
categorical_accuracy: 0.8883 - val_loss: 0.3141 - val_categorical_accuracy:
0.8882
Epoch 49/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3086 -
categorical_accuracy: 0.8884 - val_loss: 0.3122 - val_categorical_accuracy:
0.8888
Epoch 50/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3069 -
categorical_accuracy: 0.8901 - val_loss: 0.3140 - val_categorical_accuracy:
0.8878
Epoch 51/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3046 -
categorical_accuracy: 0.8894 - val_loss: 0.3115 - val_categorical_accuracy:
0.8877
Epoch 52/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3062 -
categorical_accuracy: 0.8877 - val_loss: 0.3108 - val_categorical_accuracy:
0.8869
Epoch 53/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3037 -
```

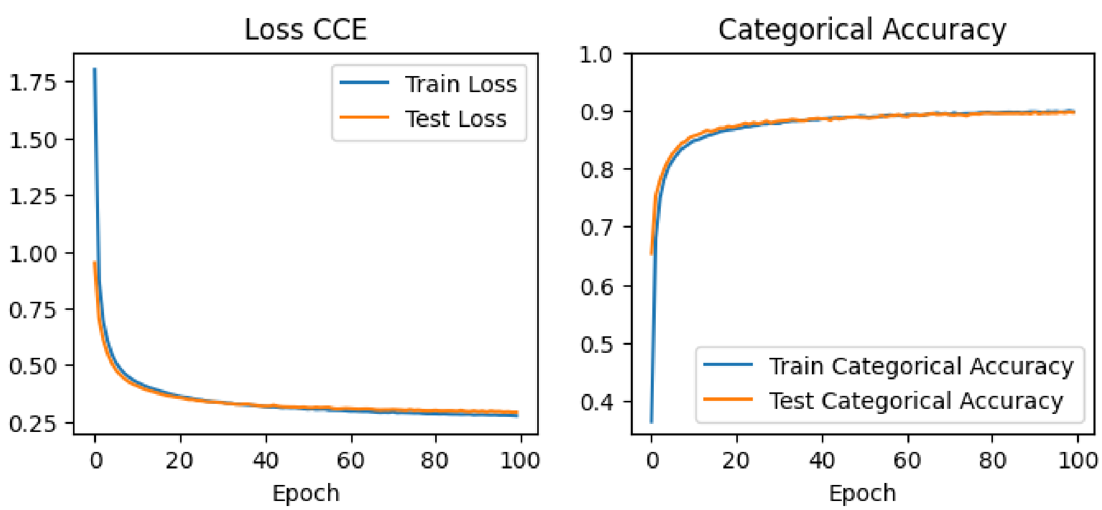
```
categorical_accuracy: 0.8889 - val_loss: 0.3130 - val_categorical_accuracy:
0.8875
Epoch 54/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3044 -
categorical_accuracy: 0.8891 - val_loss: 0.3114 - val_categorical_accuracy:
0.8883
Epoch 55/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3047 -
categorical_accuracy: 0.8892 - val_loss: 0.3124 - val_categorical_accuracy:
0.8910
Epoch 56/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3011 -
categorical_accuracy: 0.8910 - val_loss: 0.3128 - val_categorical_accuracy:
0.8888
Epoch 57/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3010 -
categorical_accuracy: 0.8918 - val_loss: 0.3074 - val_categorical_accuracy:
0.8905
Epoch 58/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3007 -
categorical_accuracy: 0.8910 - val_loss: 0.3059 - val_categorical_accuracy:
0.8910
Epoch 59/100
60/60 [==============================] - 0s 4ms/step - loss: 0.3001 -
categorical_accuracy: 0.8913 - val_loss: 0.3071 - val_categorical_accuracy:
0.8925
Epoch 60/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2985 -
categorical_accuracy: 0.8908 - val_loss: 0.3086 - val_categorical_accuracy:
0.8903
Epoch 61/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2967 -
categorical_accuracy: 0.8927 - val_loss: 0.3074 - val_categorical_accuracy:
0.8889
Epoch 62/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2970 -
categorical_accuracy: 0.8923 - val_loss: 0.3062 - val_categorical_accuracy:
0.8919
Epoch 63/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2959 -
categorical_accuracy: 0.8925 - val_loss: 0.3052 - val_categorical_accuracy:
0.8925
Epoch 64/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2953 -
categorical_accuracy: 0.8916 - val_loss: 0.3062 - val_categorical_accuracy:
0.8910
Epoch 65/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2955 -
```

```
categorical_accuracy: 0.8915 - val_loss: 0.3065 - val_categorical_accuracy:
0.8925
Epoch 66/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2936 -
categorical_accuracy: 0.8941 - val_loss: 0.3053 - val_categorical_accuracy:
0.8930
Epoch 67/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2919 -
categorical_accuracy: 0.8939 - val_loss: 0.3026 - val_categorical_accuracy:
0.8948
Epoch 68/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2904 -
categorical_accuracy: 0.8955 - val_loss: 0.3016 - val_categorical_accuracy:
0.8928
Epoch 69/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2914 -
categorical_accuracy: 0.8939 - val_loss: 0.3022 - val_categorical_accuracy:
0.8931
Epoch 70/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2905 -
categorical_accuracy: 0.8948 - val_loss: 0.3023 - val_categorical_accuracy:
0.8940
Epoch 71/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2931 -
categorical_accuracy: 0.8930 - val_loss: 0.3036 - val_categorical_accuracy:
0.8926
Epoch 72/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2890 -
categorical_accuracy: 0.8957 - val_loss: 0.3024 - val_categorical_accuracy:
0.8915
Epoch 73/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2901 -
categorical_accuracy: 0.8938 - val_loss: 0.3014 - val_categorical_accuracy:
0.8930
Epoch 74/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2892 -
categorical_accuracy: 0.8942 - val_loss: 0.2997 - val_categorical_accuracy:
0.8920
Epoch 75/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2887 -
categorical_accuracy: 0.8946 - val_loss: 0.3034 - val_categorical_accuracy:
0.8905
Epoch 76/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2877 -
categorical_accuracy: 0.8946 - val_loss: 0.2996 - val_categorical_accuracy:
0.8937
Epoch 77/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2880 -
```

```
categorical_accuracy: 0.8947 - val_loss: 0.3013 - val_categorical_accuracy:
0.8939
Epoch 78/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2858 -
categorical_accuracy: 0.8960 - val_loss: 0.2986 - val_categorical_accuracy:
0.8936
Epoch 79/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2845 -
categorical_accuracy: 0.8962 - val_loss: 0.2980 - val_categorical_accuracy:
0.8955
Epoch 80/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2847 -
categorical_accuracy: 0.8969 - val_loss: 0.2981 - val_categorical_accuracy:
0.8952
Epoch 81/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2842 -
categorical_accuracy: 0.8959 - val_loss: 0.2997 - val_categorical_accuracy:
0.8953
Epoch 82/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2836 -
categorical_accuracy: 0.8958 - val_loss: 0.2975 - val_categorical_accuracy:
0.8948
Epoch 83/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2842 -
categorical_accuracy: 0.8964 - val_loss: 0.2975 - val_categorical_accuracy:
0.8949
Epoch 84/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2830 -
categorical_accuracy: 0.8968 - val_loss: 0.2979 - val_categorical_accuracy:
0.8949
Epoch 85/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2824 -
categorical_accuracy: 0.8971 - val_loss: 0.2977 - val_categorical_accuracy:
0.8949
Epoch 86/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2825 -
categorical_accuracy: 0.8965 - val_loss: 0.2962 - val_categorical_accuracy:
0.8950
Epoch 87/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2816 -
categorical_accuracy: 0.8970 - val_loss: 0.2996 - val_categorical_accuracy:
0.8950
Epoch 88/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2820 -
categorical_accuracy: 0.8959 - val_loss: 0.2954 - val_categorical_accuracy:
0.8947
Epoch 89/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2823 -
```

```
categorical_accuracy: 0.8969 - val_loss: 0.2942 - val_categorical_accuracy:
0.8952
Epoch 90/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2796 -
categorical_accuracy: 0.8976 - val_loss: 0.2972 - val_categorical_accuracy:
0.8941
Epoch 91/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2804 -
categorical_accuracy: 0.8975 - val_loss: 0.2941 - val_categorical_accuracy:
0.8968
Epoch 92/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2806 -
categorical_accuracy: 0.8970 - val_loss: 0.2965 - val_categorical_accuracy:
0.8940
Epoch 93/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2798 -
categorical_accuracy: 0.8961 - val_loss: 0.2925 - val_categorical_accuracy:
0.8961
Epoch 94/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2791 -
categorical_accuracy: 0.8979 - val_loss: 0.2968 - val_categorical_accuracy:
0.8948
Epoch 95/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2796 -
categorical_accuracy: 0.8973 - val_loss: 0.2942 - val_categorical_accuracy:
0.8956
Epoch 96/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2780 -
categorical_accuracy: 0.8980 - val_loss: 0.2947 - val_categorical_accuracy:
0.8953
Epoch 97/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2782 -
categorical_accuracy: 0.8970 - val_loss: 0.2920 - val_categorical_accuracy:
0.8968
Epoch 98/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2777 -
categorical_accuracy: 0.8975 - val_loss: 0.2932 - val_categorical_accuracy:
0.8956
Epoch 99/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2766 -
categorical_accuracy: 0.8989 - val_loss: 0.2913 - val_categorical_accuracy:
0.8971
Epoch 100/100
60/60 [==============================] - 0s 4ms/step - loss: 0.2753 -
categorical_accuracy: 0.8977 - val_loss: 0.2914 - val_categorical_accuracy:
0.8963
```

[ ]:
```
history.history.keys()
f, ax = plt.subplots(ncols=2)
f.set_size_inches(8,3)
ax[0].plot(history.history["loss"], label="Train Loss")
ax[0].set_title("Loss CCE")
ax[0].plot(history.history["val_loss"], label="Test Loss")
# ax[0].set_yscale("log")
ax[0].set_xlabel("Epoch")
ax[0].legend()
ax[1].plot(history.history["categorical_accuracy"], label="Train Categorical␣
↪Accuracy")
ax[1].set_xlabel("Epoch")
ax[1].set_title("Categorical Accuracy")
ax[1].plot(history.history["val_categorical_accuracy"], label="Test Categorical␣
↪Accuracy")
ax[1].set_ylim(min(history.history["categorical_accuracy"])-0.02, 1)
ax[1].legend();
```

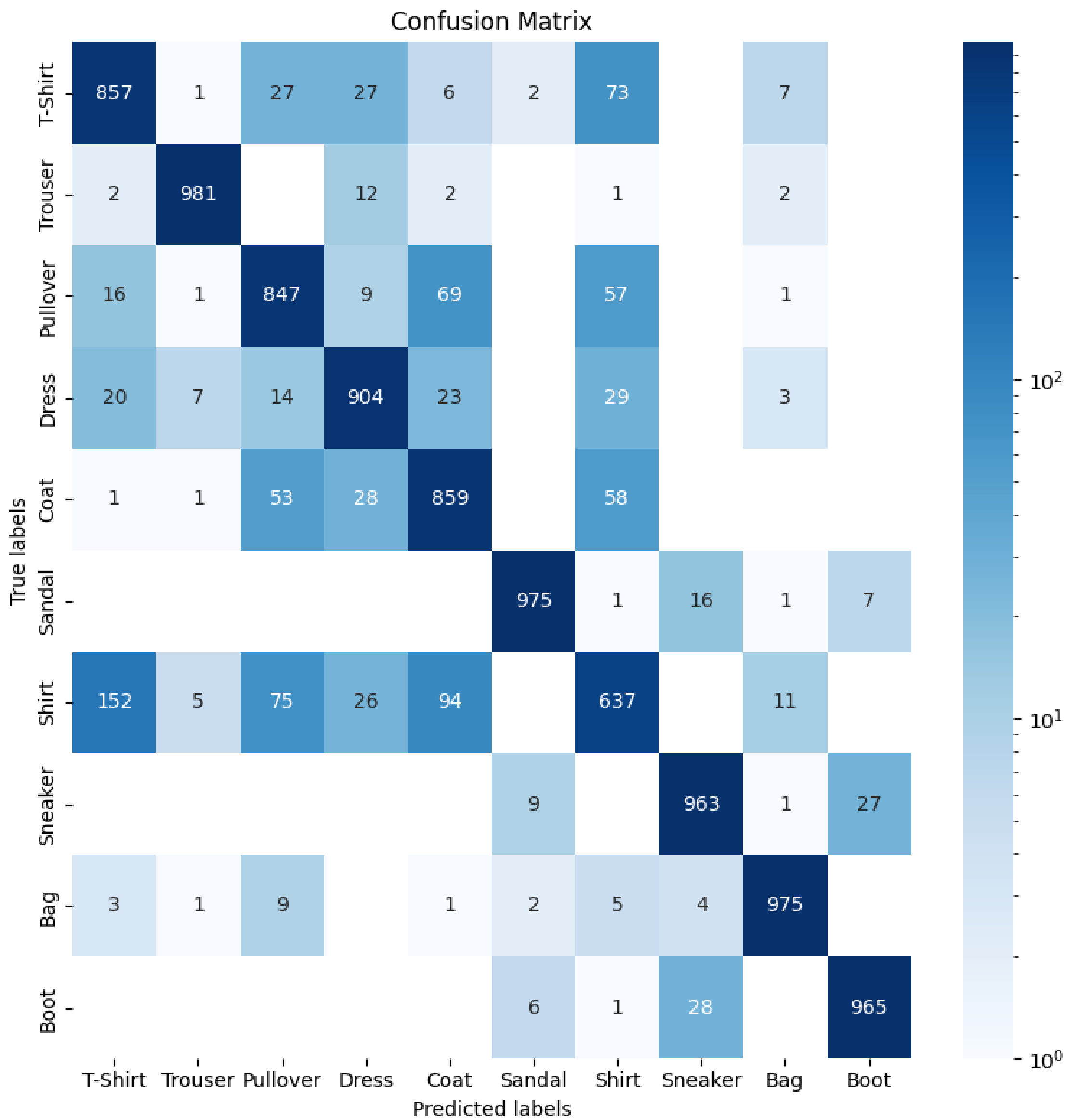


[ ]:
```
def plot_confusion_matrix(Y: np.array, pred: np.array, labels=[], savename="",␣
↪logscale=False):
    """
    Convenience function for generating a confusion Matrix

    Args:
        Y (np.array): Actual labels for the dataset (n rows, 1 column)
        pred (np.array): Predicted labels for the data (n rows, 1 column)
        labels (list of str): class labels
        savename (str, optional): File to save plot to. If none is given shows␣
↪figure.
```

```
                                    Defaults to "".

    Returns:
        confusion matrix
    """
    # Figure out predicted class -- infer from Y and pred the number of classes
    if Y.shape[1] > 1:
        Y_labels = np.zeros(Y.shape[0], dtype=int)
        pred_labels = np.zeros_like(Y_labels)
        for i in range(Y.shape[0]):
            Y_labels[i] = np.argmax(Y[i])
            pred_labels[i] = np.argmax(pred[i])
    else:
        Y_labels = Y
        pred_labels = (Y >= 0.5).astype(int)
    cm = confusion_matrix(Y_labels, pred_labels)
    f, ax = plt.subplots()
    if logscale:
        from matplotlib.colors import LogNorm, Normalize
        sns.heatmap(cm, annot=True, fmt='g', ax=ax, cmap='Blues',␣
↪norm=LogNorm())
    else:
        sns.heatmap(cm, annot=True, fmt='g', ax=ax, cmap='Blues')
    # labels, title and ticks
    ax.set_xlabel("Predicted labels")
    ax.set_ylabel("True labels")
    ax.set_title("Confusion Matrix")
    if not labels:
        labels = np.arange(max(Y.shape[1], 2))
    ax.xaxis.set_ticklabels(labels)
    ax.yaxis.set_ticklabels(labels)

    if savename != "":
        plt.savefig(savename)
        plt.close(f)
    else:
        f.set_size_inches((8,8))
        plt.tight_layout()
        plt.show()

    return cm
```

[ ]:
```
Ypred = model.predict(Xtest_norm)
plot_confusion_matrix(Ytest_OHE, Ypred, labels=mnist_labels, logscale=True)
```

```
313/313 [==============================] - 0s 779us/step
```




```
[ ]: array([[857,   1,  27,  27,   6,   2,  73,   0,   7,   0],
              [  2, 981,   0,  12,   2,   0,   1,   0,   2,   0],
              [ 16,   1, 847,   9,  69,   0,  57,   0,   1,   0],
              [ 20,   7,  14, 904,  23,   0,  29,   0,   3,   0],
              [  1,   1,  53,  28, 859,   0,  58,   0,   0,   0],
              [  0,   0,   0,   0,   0, 975,   1,  16,   1,   7],
              [152,   5,  75,  26,  94,   0, 637,   0,  11,   0],
              [  0,   0,   0,   0,   0,   9,   0, 963,   1,  27],
              [  3,   1,   9,   0,   1,   2,   5,   4, 975,   0],
              [  0,   0,   0,   0,   0,   6,   1,  28,   0, 965]])
```

The cases it gets wrong makes sense, as most of the confusion is between shirts and shirt-like objects. Pants are predicted well, and the little confusion with footware is between different types of footware.