**Clothing**

# TensorFlow and tf.keras

import tensorflow as tf

# Helper libraries

import numpy as np

import matplotlib.pyplot as plt

print(tf.\_\_version\_\_)

fashion\_mnist = tf.keras.datasets.fashion\_mnist

(train\_images, train\_labels), (test\_images, test\_labels) = fashion\_mnist.load\_data()

class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

train\_images.shape

train\_images.shape

len(train\_labels)

train\_labels

test\_images.shape

len(test\_labels)

#preprocessing

plt.figure()

plt.imshow(train\_images[0])

plt.colorbar()

plt.grid(False)

plt.show()

train\_images = train\_images / 255.0

test\_images = test\_images / 255.0

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

for i in range(25):

plt.subplot(5,5,i+1)

plt.xticks([])

plt.yticks([])

plt.grid(False)

plt.imshow(train\_images[i], cmap=plt.cm.binary)

plt.xlabel(class\_names[train\_labels[i]])

plt.show()

model = tf.keras.Sequential([

tf.keras.layers.Flatten(input\_shape=(28, 28)),

tf.keras.layers.Dense(128, activation='relu'),

tf.keras.layers.Dense(10)

])

model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=['accuracy'])

model.fit(train\_images, train\_labels, epochs=30)

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels, verbose=2)

print('\nTest accuracy:', test\_acc)

probability\_model = tf.keras.Sequential([model,

tf.keras.layers.Softmax()])

predictions = probability\_model.predict(test\_images)

predictions[0]

np.argmax(predictions[0])

test\_labels[0]

def plot\_image(i, predictions\_array, true\_label, img):

true\_label, img = true\_label[i], img[i]

plt.grid(False)

plt.xticks([])

plt.yticks([])

plt.imshow(img, cmap=plt.cm.binary)

predicted\_label = np.argmax(predictions\_array)

if predicted\_label == true\_label:

color = 'blue'

else:

color = 'red'

plt.xlabel("{} {:2.0f}% ({})".format(class\_names[predicted\_label],

100\*np.max(predictions\_array),

class\_names[true\_label]),

color=color)

def plot\_value\_array(i, predictions\_array, true\_label):

true\_label = true\_label[i]

plt.grid(False)

plt.xticks(range(10))

plt.yticks([])

thisplot = plt.bar(range(10), predictions\_array, color="#777777")

plt.ylim([0, 1])

predicted\_label = np.argmax(predictions\_array)

thisplot[predicted\_label].set\_color('red')

thisplot[true\_label].set\_color('blue')

i = 0

plt.figure(figsize=(6,3))

plt.subplot(1,2,1)

plot\_image(i, predictions[i], test\_labels, test\_images)

plt.subplot(1,2,2)

plot\_value\_array(i, predictions[i], test\_labels)

plt.show()

i = 12

plt.figure(figsize=(6,3))

plt.subplot(1,2,1)

plot\_image(i, predictions[i], test\_labels, test\_images)

plt.subplot(1,2,2)

plot\_value\_array(i, predictions[i], test\_labels)

plt.show()

# Plot the first X test images, their predicted labels, and the true labels.

# Color correct predictions in blue and incorrect predictions in red.

num\_rows = 5

num\_cols = 3

num\_images = num\_rows\*num\_cols

plt.figure(figsize=(2\*2\*num\_cols, 2\*num\_rows))

for i in range(num\_images):

plt.subplot(num\_rows, 2\*num\_cols, 2\*i+1)

plot\_image(i, predictions[i], test\_labels, test\_images)

plt.subplot(num\_rows, 2\*num\_cols, 2\*i+2)

plot\_value\_array(i, predictions[i], test\_labels)

plt.tight\_layout()

plt.show()

# Plot the first X test images, their predicted labels, and the true labels.

# Color correct predictions in blue and incorrect predictions in red.

num\_rows = 5

num\_cols = 3

num\_images = num\_rows\*num\_cols

plt.figure(figsize=(2\*2\*num\_cols, 2\*num\_rows))

for i in range(num\_images):

plt.subplot(num\_rows, 2\*num\_cols, 2\*i+1)

plot\_image(i, predictions[i], test\_labels, test\_images)

plt.subplot(num\_rows, 2\*num\_cols, 2\*i+2)

plot\_value\_array(i, predictions[i], test\_labels)

plt.tight\_layout()

plt.show()

# Grab an image from the test dataset.

img = test\_images[1]

print(img.shape)

# Add the image to a batch where it's the only member.

img = (np.expand\_dims(img,0))\

print(img.shape)

predictions\_single = probability\_model.predict(img)

print(predictions\_single)

plot\_value\_array(1, predictions\_single[0], test\_labels)

\_ = plt.xticks(range(10), class\_names, rotation=45)

plt.show()

np.argmax(predictions\_single[0])

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.datasets import fashion\_mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from tensorflow.keras.utils import to\_categorical

# Load the MNIST Fashion dataset

(x\_train, y\_train), (x\_test, y\_test) = fashion\_mnist.load\_data()

# Preprocess the data

x\_train = x\_train.reshape(x\_train.shape[0], 28, 28, 1) # Reshape to 28x28 grayscale images

x\_test = x\_test.reshape(x\_test.shape[0], 28, 28, 1)

x\_train = x\_train.astype('float32') / 255 # Normalize pixel values to range [0, 1]

x\_test = x\_test.astype('float32') / 255

y\_train = to\_categorical(y\_train, num\_classes=10) # Convert labels to one-hot encoded vectors

y\_test = to\_categorical(y\_test, num\_classes=10)

# Define the model architecture

model = Sequential([

Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(28, 28, 1)),

MaxPooling2D(pool\_size=(2, 2)),

Conv2D(64, kernel\_size=(3, 3), activation='relu'),

MaxPooling2D(pool\_size=(2, 2)),

Flatten(),

Dense(128, activation='relu'),

Dropout(0.5),

Dense(10, activation='softmax')

])

# Compile the model

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model

model.fit(x\_train, y\_train, batch\_size=128, epochs=10, validation\_data=(x\_test, y\_test))

# Evaluate the model on the testing set

loss, accuracy = model.evaluate(x\_test, y\_test)

print("Testing loss:", loss)

print("Testing accuracy:", accuracy)

# Define the model architecture

model = Sequential([

Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(28, 28, 1)),

MaxPooling2D(pool\_size=(2, 2)),

Conv2D(64, kernel\_size=(3, 3), activation='relu'),

MaxPooling2D(pool\_size=(2, 2)),

Flatten(),

Dense(128, activation='relu'),

Dropout(0.5),

Dense(10, activation='softmax')

])

# Compile the model

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model

model.fit(x\_train, y\_train, batch\_size=128, epochs=10, validation\_data=(x\_test, y\_test))