Seeding for reproducibility

```
# Set seeds for reproducibility
import random
random.seed(0)

import numpy as np
np.random.seed(0)

import tensorflow as tf
tf.random.set_seed(0)
```

Observation:This is for getting the same result just like random_state.Whenver everyone uses the same number we will get the same result .Machine learning is all about randomness

Importing the dependencies

```
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

Data Curation

```
# Load and prepare the Fashion MNIST dataset
fashion_mnist = datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

Data Processing

```
type(train_images)
```

```
numpy.ndarray
```

```
type(train_labels)
```

```
numpy.ndarray
```

```
print(len(train_images))
```

```
60000
```

```
print(len(train_labels))
```

```
60000
```

```
print(len(test_images))
```

```
10000
```

```
print(train_images[0].shape)
```

```
(28, 28)
```

```
print(type(train_images[0]))
```

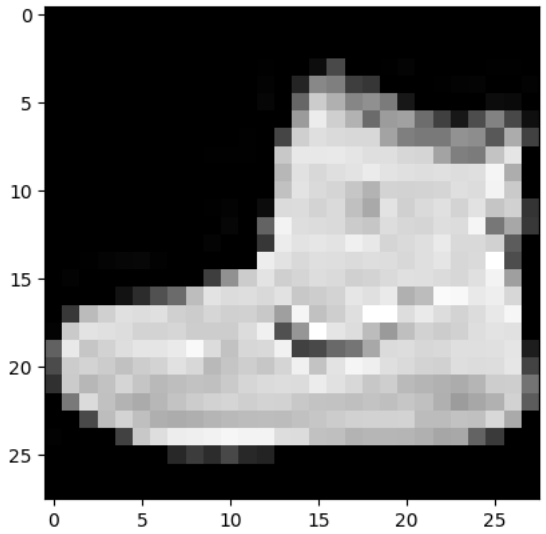
```
<class 'numpy.ndarray'>
```

```
print(train_images[0])
```

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0  13  73   0
```

```
   0   1   4   0   0   0   0   1   1   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   3   0  36 136 127  62
  54   0   0   0   1   3   4   0   0   3]
[  0   0   0   0   0   0   0   0   0   0   0   0   6   0 102 204 176 134
 144 123  23   0   0   0   0  12  10   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0 155 236 207 178
 107 156 161 109  64  23  77 130  72  15]
[  0   0   0   0   0   0   0   0   0   0   0   1   0  69 207 223 218 216
 216 163 127 121 122 146 141  88 172  66]
[  0   0   0   0   0   0   0   0   0   1   1   1   0 200 232 232 233 229
 223 223 215 213 164 127 123 196 229   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0 183 225 216 223 228
 235 227 224 222 224 221 223 245 173   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0 193 228 218 213 198
 180 212 210 211 213 223 220 243 202   0]
[  0   0   0   0   0   0   0   0   0   1   3   0  12 219 220 212 218 192
 169 227 208 218 224 212 226 197 209  52]
[  0   0   0   0   0   0   0   0   0   0   6   0  99 244 222 220 218 203
 198 221 215 213 222 220 245 119 167  56]
[  0   0   0   0   0   0   0   0   0   4   0   0  55 236 228 230 228 240
 232 213 218 223 234 217 217 209  92   0]
[  0   0   1   4   6   7   2   0   0   0   0   0 237 226 217 223 222 219
 222 221 216 223 229 215 218 255  77   0]
[  0   3   0   0   0   0   0   0   0  62 145 204 228 207 213 221 218 208
 211 218 224 223 219 215 224 244 159   0]
[  0   0   0   0  18  44  82 107 189 228 220 222 217 226 200 205 211 230
 224 234 176 188 250 248 233 238 215   0]
[  0  57 187 208 224 221 224 208 204 214 208 209 200 159 245 193 206 223
 255 255 221 234 221 211 220 232 246   0]
[  3 202 228 224 221 211 211 214 205 205 205 220 240  80 150 255 229 221
 188 154 191 210 204 209 222 228 225   0]
[ 98 233 198 210 222 229 229 234 249 220 194 215 217 241  65  73 106 117
 168 219 221 215 217 223 223 224 229  29]
[ 75 204 212 204 193 205 211 225 216 185 197 206 198 213 240 195 227 245
 239 223 218 212 209 222 220 221 230  67]
[ 48 203 183 194 213 197 185 190 194 192 202 214 219 221 220 236 225 216
 199 206 186 181 177 172 181 205 206 115]
[  0 122 219 193 179 171 183 196 204 210 213 207 211 210 200 196 194 191
 195 191 198 192 176 156 167 177 210  92]
[  0   0  74 189 212 191 175 172 175 181 185 188 189 188 193 198 204 209
 210 210 211 188 188 194 192 216 170   0]
[  2   0   0   0  66 200 222 237 239 242 246 243 244 221 220 193 191 179
 182 182 181 176 166 168  99  58   0   0]
[  0   0   0   0   0   0   0  40  61  44  72  41  35   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]]
```

```
# Display an image from the dataset
plt.imshow(train_images[0], cmap='gray')
plt.show()
```




```
###To save some images
plt.imsave("fashion_sample.jpg", test_images[5].squeeze(), cmap='gray')
```

```
print(train_labels[0])
```

```
9
```

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

```
# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
```

Observation:We highest value is 255(white) while the lowest is 0 (dark).We must divide by 255 inorder to get values between 0 and 1

```
print(train_images[0])
```

```
  0.85098039 0.88627451 0.78431373 0.80392157 0.82745098 0.90196078
  0.87843137 0.91764706 0.69019608 0.7372549  0.98039216 0.97254902
  0.91372549 0.93333333 0.84313725 0.        ]
 [0.         0.22352941 0.73333333 0.81568627 0.87843137 0.86666667
  0.87843137 0.81568627 0.8        0.83921569 0.81568627 0.81960784
  0.78431373 0.62352941 0.96078431 0.75686275 0.80784314 0.8745098
  1.         1.         0.86666667 0.91764706 0.86666667 0.82745098
  0.8627451  0.90980392 0.96470588 0.        ]
 [0.01176471 0.79215686 0.89411765 0.87843137 0.86666667 0.82745098
  0.82745098 0.83921569 0.80392157 0.80392157 0.80392157 0.8627451
  0.94117647 0.31372549 0.58823529 1.         0.89803922 0.86666667
  0.7372549  0.60392157 0.74901961 0.82352941 0.8        0.81960784
  0.87058824 0.89411765 0.88235294 0.        ]
 [0.38431373 0.91372549 0.77647059 0.82352941 0.87058824 0.89803922
  0.89803922 0.91764706 0.97647059 0.8627451  0.76078431 0.84313725
  0.85098039 0.94509804 0.25490196 0.28627451 0.41568627 0.45882353
  0.65882353 0.85882353 0.86666667 0.84313725 0.85098039 0.8745098
  0.8745098  0.87843137 0.89803922 0.11372549]
 [0.29411765 0.8        0.83137255 0.8        0.75686275 0.80392157
  0.82745098 0.88235294 0.84705882 0.7254902  0.77254902 0.80784314
  0.77647059 0.83529412 0.94117647 0.76470588 0.89019608 0.96078431
  0.9372549  0.8745098  0.85490196 0.83137255 0.81960784 0.87058824
  0.8627451  0.86666667 0.90196078 0.2627451 ]
 [0.18823529 0.79607843 0.71764706 0.76078431 0.83529412 0.77254902
  0.7254902  0.74509804 0.76078431 0.75294118 0.79215686 0.83921569
  0.85882353 0.86666667 0.8627451  0.9254902  0.88235294 0.84705882
  0.78039216 0.80784314 0.72941176 0.70980392 0.69411765 0.6745098
  0.70980392 0.80392157 0.80784314 0.45098039]
 [0.         0.47843137 0.85882353 0.75686275 0.70196078 0.67058824
  0.71764706 0.76862745 0.8        0.82352941 0.83529412 0.81176471
  0.82745098 0.82352941 0.78431373 0.76862745 0.76078431 0.74901961
  0.76470588 0.74901961 0.77647059 0.75294118 0.69019608 0.61176471
  0.65490196 0.69411765 0.82352941 0.36078431]
 [0.         0.         0.29019608 0.74117647 0.83137255 0.74901961
  0.68627451 0.6745098  0.68627451 0.70980392 0.7254902  0.7372549
  0.74117647 0.7372549  0.75686275 0.77647059 0.8        0.81960784
  0.82352941 0.82352941 0.82745098 0.7372549  0.7372549  0.76078431
  0.75294118 0.84705882 0.66666667 0.        ]
 [0.00784314 0.         0.         0.         0.25882353 0.78431373
  0.87058824 0.92941176 0.9372549  0.94901961 0.96470588 0.95294118
  0.95686275 0.86666667 0.8627451  0.75686275 0.74901961 0.70196078
  0.71372549 0.71372549 0.70980392 0.69019608 0.65098039 0.65882353
  0.38823529 0.22745098 0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.15686275 0.23921569 0.17254902 0.28235294 0.16078431
  0.1372549  0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]]
```

```
# Reshape images to specify that it's a single channel (grayscale)
train_images = train_images.reshape((train_images.shape[0], 28, 28, 1))
test_images = test_images.reshape((test_images.shape[0], 28, 28, 1))
```

```
train_images.shape
```

```
(60000, 28, 28, 1)
```

```
test_images.shape
```

```
(10000, 28, 28, 1)
```

Convolutional Neural Network

```
# Build the convolutional base
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Add Dense layers on top
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

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

Whenever we usefrom_logit=True we cant use the activation energy in the output

Model Training

```
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 65s 34ms/step - accuracy: 0.7489 - loss: 0.6951 - val_accuracy: 0.8695 - val_loss: 0.3626
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 60s 32ms/step - accuracy: 0.8779 - loss: 0.3357 - val_accuracy: 0.8835 - val_loss: 0.3260
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 83s 32ms/step - accuracy: 0.8975 - loss: 0.2822 - val_accuracy: 0.8895 - val_loss: 0.3024
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 81s 32ms/step - accuracy: 0.9085 - loss: 0.2481 - val_accuracy: 0.8976 - val_loss: 0.2817
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 60s 32ms/step - accuracy: 0.9184 - loss: 0.2195 - val_accuracy: 0.8994 - val_loss: 0.2864
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 62s 33ms/step - accuracy: 0.9265 - loss: 0.1969 - val_accuracy: 0.9002 - val_loss: 0.2927
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 80s 32ms/step - accuracy: 0.9340 - loss: 0.1760 - val_accuracy: 0.9014 - val_loss: 0.2928
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 60s 32ms/step - accuracy: 0.9412 - loss: 0.1574 - val_accuracy: 0.9024 - val_loss: 0.2948
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 60s 32ms/step - accuracy: 0.9447 - loss: 0.1451 - val_accuracy: 0.8977 - val_loss: 0.3261
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 62s 33ms/step - accuracy: 0.9509 - loss: 0.1301 - val_accuracy: 0.9029 - val_loss: 0.3283
```

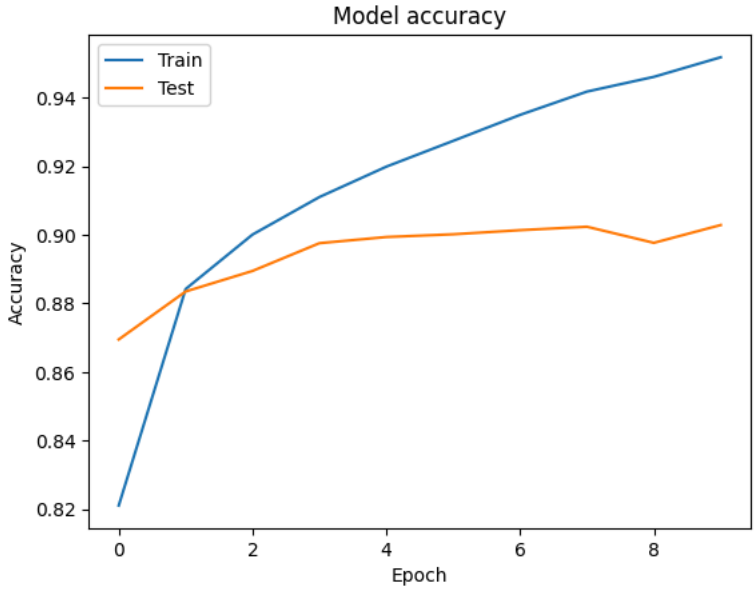
Model Evaluation

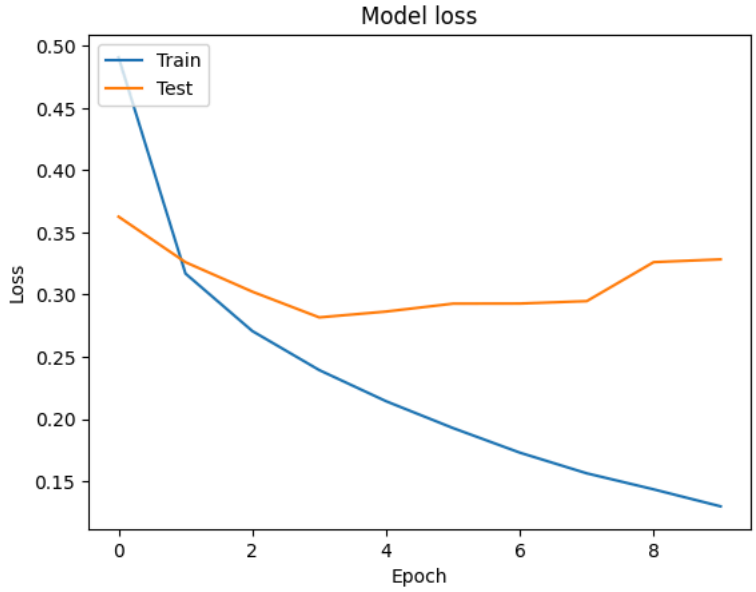
```
# Evaluate the model
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
print('\nTest accuracy:', test_acc)
```

```
313/313 - 3s - 8ms/step - accuracy: 0.9029 - loss: 0.3283

Test accuracy: 0.902899980545044
```

```
# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
```

```
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```






```
model.save('trained_fashion_mnist_model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is
```

Start coding or generate with AI.