Generate | 10 random numbers using numpy | Close

```python
from tensorflow.keras.datasets.fashion_mnist import load_data
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Input, Flatten, Dense, Conv2D, MaxPooling2D, Dropout
from tensorflow.keras.models import Model
import random
```

Generate | a slider using jupyter widgets | Close

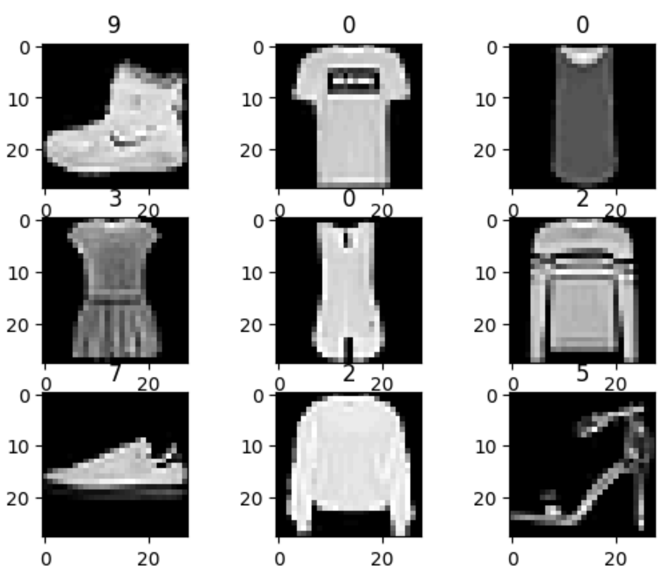
```python
def display_img(img_set, title_set):
  n = len(title_set)
  for i in range(n):
    plt.subplot(3, 3, i + 1)
    plt.imshow(img_set[i], cmap = 'gray')
    plt.title(title_set[i])
  plt.show()
  plt.close()


# Load dataset
(trainX, trainY), (testX, testY) = load_data()

# Investigate loaded data
print(f'trainX.shape: {trainX.shape}, trainY.shape: {trainY.shape}, testX.shape: {testX.shape}, testY.shape: {testY.shape}')
print(f'trainX.dtype: {trainX.dtype}, trainY.dtype: {trainY.dtype}, testX.dtype: {testX.dtype}, testY.dtype: {testY.dtype}')
print(f'trainX.Range: {trainX.min()} to {trainX.max()}, testX.Range: {testX.min()} to {testX.max()}')

# Display sample images
display_img(trainX[:9], trainY[:9])
```

```
trainX.shape: (60000, 28, 28), trainY.shape: (60000,), testX.shape: (10000, 28, 28), testY.shape: (10000,)
trainX.dtype: uint8, trainY.dtype: uint8, testX.dtype: uint8, testY.dtype: uint8
trainX.Range: 0 to 255, testX.Range: 0 to 255
```




```python
# Expand dimensions for grayscale images
trainX = np.expand_dims(trainX, axis=-1)
testX = np.expand_dims(testX, axis=-1)

# One-hot encode labels
trainY = to_categorical(trainY, num_classes=10)
testY = to_categorical(testY, num_classes=10)

# Investigate updated y
print(f'trainY.shape: {trainY.shape}, testY.shape: {testY.shape}')
print(f'trainY.dtype: {trainY.dtype}, testY.dtype: {testY.dtype}')
print(f'One-hot encoded labels (first 5): \n{trainY[:5]}')
```

```
# Normalize pixel values to the range [0, 1]
trainX = trainX.astype('float32') / 255
testX = testX.astype('float32') / 255
```

```
trainY.shape: (60000, 10), testY.shape: (10000, 10)
trainY.dtype: float64, testY.dtype: float64
One-hot encoded labels (first 5):
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

```
# Fully connected model
fc_inputs = Input((28, 28, 1), name='FC_InputLayer')
x = Flatten()(fc_inputs)
x = Dense(512, activation='relu')(x)
x = Dense(4, activation='relu')(x)
x = Dense(8, activation='relu')(x)
x = Dense(16, activation='relu')(x)
x = Dense(8, activation='relu')(x)
x = Dense(4, activation='relu')(x)
fc_outputs = Dense(10, name='FC_OutputLayer', activation='softmax')(x)
fc_model = Model(fc_inputs, fc_outputs, name='Fully-Connected-Classifier')
fc_model.summary()
```

Model: "Fully-Connected-Classifier"

Layer (type)	Output Shape	Param #
FC_InputLayer (InputLayer)	(None, 28, 28, 1)	0
flatten_2 (Flatten)	(None, 784)	0
dense_7 (Dense)	(None, 512)	401,920
dense_8 (Dense)	(None, 4)	2,052
dense_9 (Dense)	(None, 8)	40
dense_10 (Dense)	(None, 16)	144
dense_11 (Dense)	(None, 8)	136
dense_12 (Dense)	(None, 4)	36
FC_OutputLayer (Dense)	(None, 10)	50

Total params: 404,378 (1.54 MB)
Trainable params: 404,378 (1.54 MB)

```
# Compile model
fc_model.compile(optimizer="rmsprop", loss='categorical_crossentropy', metrics=['accuracy'])

# Train fully connected model
print("\nTraining Fully Connected Model...\n")
fc_model.fit(trainX, trainY, batch_size=128, validation_split=0.1, epochs=10)
```

```
Training Fully Connected Model...

Epoch 1/10
422/422 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - accuracy: 0.2297 - loss: 1.8332 - val_accuracy: 0.4805 - val_loss: 1.4550
Epoch 2/10
422/422 ━━━━━━━━━━━━━━━━━━━━ 5s 12ms/step - accuracy: 0.4756 - loss: 1.3524 - val_accuracy: 0.6017 - val_loss: 1.1499
Epoch 3/10
422/422 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.6207 - loss: 1.0500 - val_accuracy: 0.6480 - val_loss: 0.9253
Epoch 4/10
422/422 ━━━━━━━━━━━━━━━━━━━━ 6s 13ms/step - accuracy: 0.6395 - loss: 0.8979 - val_accuracy: 0.6847 - val_loss: 0.8246
Epoch 5/10
422/422 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - accuracy: 0.7213 - loss: 0.7744 - val_accuracy: 0.7425 - val_loss: 0.7678
Epoch 6/10
422/422 ━━━━━━━━━━━━━━━━━━━━ 7s 13ms/step - accuracy: 0.7959 - loss: 0.6457 - val_accuracy: 0.8370 - val_loss: 0.6045
Epoch 7/10
422/422 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.8489 - loss: 0.5509 - val_accuracy: 0.8435 - val_loss: 0.5310
Epoch 8/10
422/422 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.8627 - loss: 0.4759 - val_accuracy: 0.8535 - val_loss: 0.4972
Epoch 9/10
```

```
422/422 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.8688 - loss: 0.4269 - val_accuracy: 0.8600 - val_loss: 0.4533
Epoch 10/10
422/422 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.8759 - loss: 0.3870 - val_accuracy: 0.8682 - val_loss: 0.4201
<keras.src.callbacks.history.History at 0x7c3d35c99420>
```

```
# Evaluate FCNN Model
print("\nEvaluating Fully Connected Neural Network (FCNN)...\n")
fc_score = fc_model.evaluate(testX, testY)

# Predict using FCNN Model
fc_predictions = fc_model.predict(testX)
```

```
# Compare Predictions for the First 10 Test Samples
print('OriginalY   FCNN_PredictedY')
print('=========   ===============')

for i in range(10):
    true_label = np.argmax(testY[i])
    fc_pred = np.argmax(fc_predictions[i])
    print(f'{true_label:<10} {fc_pred}')
```

```
Evaluating Fully Connected Neural Network (FCNN)...

313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8659 - loss: 0.4272
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
OriginalY   FCNN_PredictedY
=========   ===============
9          9
2          2
1          1
1          1
6          6
1          1
4          4
6          6
5          5
7          7
```

```
# Define CNN Model
cnn_inputs = Input((28, 28, 1), name='InputLayer')  # Change input shape to (28, 28, 1)

x = Conv2D(filters=8, kernel_size=(5, 5), padding='same', activation='relu')(cnn_inputs)
x = Conv2D(filters=8, kernel_size=(5, 5), padding='same', activation='relu')(x)
x = MaxPooling2D()(x)  # Downsampling

x = Conv2D(filters=16, kernel_size=(5, 5), activation='relu')(x)
x = Conv2D(filters=16, kernel_size=(5, 5), activation='relu')(x)
x = Conv2D(filters=16, kernel_size=(5, 5), strides=(2, 2), activation='relu')(x)  # Downsampling with strides

x = Flatten()(x)  # Flatten feature maps
x = Dense(8, activation='relu')(x)  # Dense layer with 8 neurons

cnn_outputs = Dense(10, name='OutputLayer', activation='softmax')(x)  # Output layer for 10 classes

# Compile and Summarize CNN Model
cnn_model = Model(cnn_inputs, cnn_outputs, name='CNN')
cnn_model.summary()
```

```
Model: "CNN"
```

Layer (type)	Output Shape	Param #
InputLayer (InputLayer)	(None, 28, 28, 1)	0
conv2d_12 (Conv2D)	(None, 28, 28, 8)	208
conv2d_13 (Conv2D)	(None, 28, 28, 8)	1,608
max_pooling2d_8 (MaxPooling2D)	(None, 14, 14, 8)	0
conv2d_14 (Conv2D)	(None, 10, 10, 16)	3,216
conv2d_15 (Conv2D)	(None, 6, 6, 16)	6,416
conv2d_16 (Conv2D)	(None, 1, 1, 16)	6,416
flatten_6 (Flatten)	(None, 16)	0
dense_17 (Dense)	(None, 8)	136
OutputLayer (Dense)	(None, 10)	90

Total params: 18,090 (70.66 KB)
Trainable params: 18,090 (70.66 KB)

```
# Compile models
cnn_model.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy'])


# Train CNN model
print("\nTraining CNN Model...\n")
cnn_model.fit(trainX, trainY, batch_size=128, validation_split=0.1, epochs=10)
```

```
Training CNN Model...

Epoch 1/10
422/422 ━━━━━━━━━━━━━━━━━━━━ 151s 218ms/step - accuracy: 0.4299 - loss: 1.5813 - val_accuracy: 0.7672 - val_loss: 0.6311
Epoch 2/10
422/422 ━━━━━━━━━━━━━━━━━━━━ 92s 218ms/step - accuracy: 0.7733 - loss: 0.6078 - val_accuracy: 0.7998 - val_loss: 0.5382
Epoch 3/10
422/422 ━━━━━━━━━━━━━━━━━━━━ 92s 219ms/step - accuracy: 0.8064 - loss: 0.5188 - val_accuracy: 0.8263 - val_loss: 0.4692
Epoch 4/10
422/422 ━━━━━━━━━━━━━━━━━━━━ 93s 220ms/step - accuracy: 0.8293 - loss: 0.4640 - val_accuracy: 0.8393 - val_loss: 0.4457
Epoch 5/10
422/422 ━━━━━━━━━━━━━━━━━━━━ 141s 217ms/step - accuracy: 0.8427 - loss: 0.4322 - val_accuracy: 0.8408 - val_loss: 0.4321
Epoch 6/10
422/422 ━━━━━━━━━━━━━━━━━━━━ 143s 220ms/step - accuracy: 0.8557 - loss: 0.4036 - val_accuracy: 0.8597 - val_loss: 0.3890
Epoch 7/10
422/422 ━━━━━━━━━━━━━━━━━━━━ 142s 222ms/step - accuracy: 0.8663 - loss: 0.3741 - val_accuracy: 0.8593 - val_loss: 0.3741
Epoch 8/10
422/422 ━━━━━━━━━━━━━━━━━━━━ 93s 221ms/step - accuracy: 0.8686 - loss: 0.3670 - val_accuracy: 0.8677 - val_loss: 0.3629
Epoch 9/10
422/422 ━━━━━━━━━━━━━━━━━━━━ 142s 222ms/step - accuracy: 0.8749 - loss: 0.3443 - val_accuracy: 0.8755 - val_loss: 0.3455
Epoch 10/10
422/422 ━━━━━━━━━━━━━━━━━━━━ 141s 219ms/step - accuracy: 0.8835 - loss: 0.3319 - val_accuracy: 0.8730 - val_loss: 0.3701
<keras.src.callbacks.history.History at 0x7c3d0f7851e0>
```

```
# Evaluate CNN Model
print("\nEvaluating Convolutional Neural Network (CNN)...\n")
cnn_score = cnn_model.evaluate(testX, testY)

# Predict using CNN Model
cnn_predictions = cnn_model.predict(testX)



# Compare Predictions for the First 10 Test Samples
print('OriginalY   CNN_PredictedY')
print('=========   ===============')

for i in range(10):
    true_label = np.argmax(testY[i])
    cnn_pred = np.argmax(cnn_predictions[i])
    print(f'{true_label:<10} {cnn_pred}')
```

```
Evaluating Convolutional Neural Network (CNN)...
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 14ms/step - accuracy: 0.8669 - loss: 0.3928
313/313 ━━━━━━━━━━━━━━━━━━━━ 7s 21ms/step
OriginalY   CNN_PredictedY
=========   ==============
9           9
2           2
1           1
1           1
6           6
1           1
4           4
6           6
5           5
7           7
```