Construct a deep forward neural network

```
# Import libraries for use
import tensorflow as tf
from tensorflow import keras
from keras.utils import to_categorical
from keras import models
from keras import layers
from keras.utils import plot_model
import matplotlib.pyplot as plt

print(tf.__version__)
print(keras.__version__)
```

```
# Mount my Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
!jupyter nbconvert --to html "/content/drive/MyDrive/SIT744 - Deep Learning/Assig
```

```
# Load the Fashion-MNIST dataset
(train_images, train_labels), (test_images, test_labels) = keras.datasets.fashion
```

```
# Print shape of training and test data
print(f"Shape of training images: {train_images.shape} ")
print(f"Shape of training labels: {train_labels.shape} ")
print(f"Shape of test images: {test_images.shape} ")
print(f"Shape of test labels: {test_labels.shape} ")
```

```
Shape of training images: (60000, 28, 28)
Shape of training labels: (60000,)
Shape of test images: (10000, 28, 28)
Shape of test labels: (10000,)
```

```
# # Print size of training and test data
print(f"Size of training images: {train_images.nbytes / (1024 * 1024)} MB")
print(f"Size of training labels: {train_labels.nbytes / (1024 * 1024)} MB")
print(f"Size of test images: {test_images.nbytes / (1024 * 1024)} MB")
print(f"Size of test labels: {test_labels.nbytes / (1024 * 1024)} MB")
```

```
Size of training images: 44.86083984375 MB
Size of training labels: 0.057220458984375 MB
Size of test images: 7.476806640625 MB
Size of test labels: 0.0095367431640625 MB
```

```
# Create a list of class names
classes = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shir

# View the content of classes
classes
```

```
# Create a figure with 10 subplots (one for each class)
fig, ax = plt.subplots(nrows=10, ncols=10, figsize=(15,15))
fig.subplots_adjust(hspace=1.0, wspace=1.0)

# Loop through each class. Select 10 images of each class to display
for i in range(len(classes)):
    # Select images with the current class label
    images = train_images[train_labels == i]
    # Loop through each image and display
    for j in range(10):
        # Set the current subplot
        ax[i,j].imshow(images[j], cmap='gray')
        ax[i,j].set_title(classes[i])
        ax[i,j].axis('off')

# Show the plot and save the plot image
plt.plot()
# plt.savefig('300054233-assignment1-fashion-mnist-10-classes.png', bbox_inches='
```

```
[]
```


```
# Print the shape of a training image
train_images[0].shape
```

```
(28, 28)
```

```
# Print the matrix of a training image
train_images[0]
```

```
# Reshape training images and test images into 1D
train_images = train_images.reshape((60000, 28 * 28))
test_images = test_images.reshape((10000, 28 * 28))
```

```
# Normalise training images and test images to float32 datatype
train_images = train_images.astype('float32') / 255
test_images = test_images.astype('float32') / 255
```

```
# Encode trainning labels and test labels using one-hot encoding
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

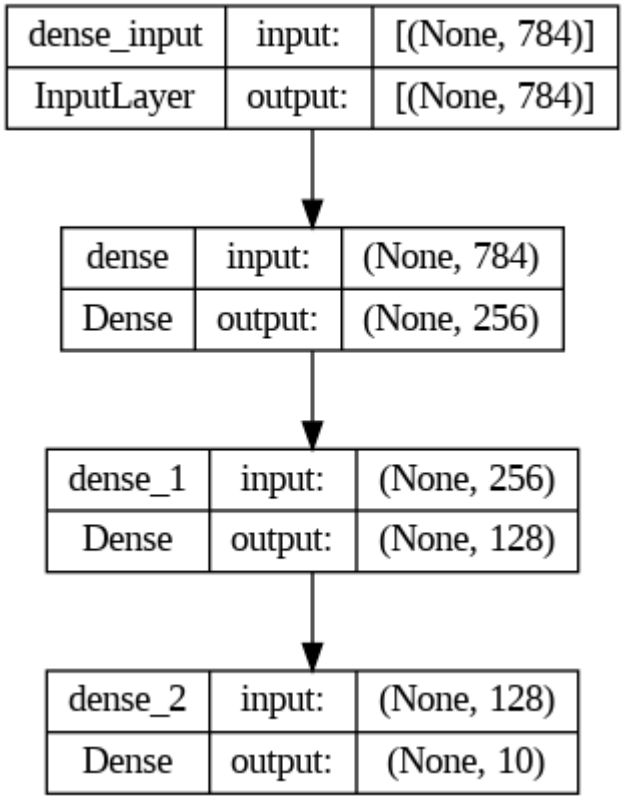
```
# Print a vector of training label
train_labels[0]
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 1.], dtype=float32)
```

Setting up a model for training

```
# Create model with 3 dense layers
neuralNetwork = models.Sequential([
    layers.Dense(256, activation='relu', input_shape=(28 * 28, )),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')])
```

```
# Plot the shape of the model
plot_model(neuralNetwork, show_shapes=True)
```



Fitting the model

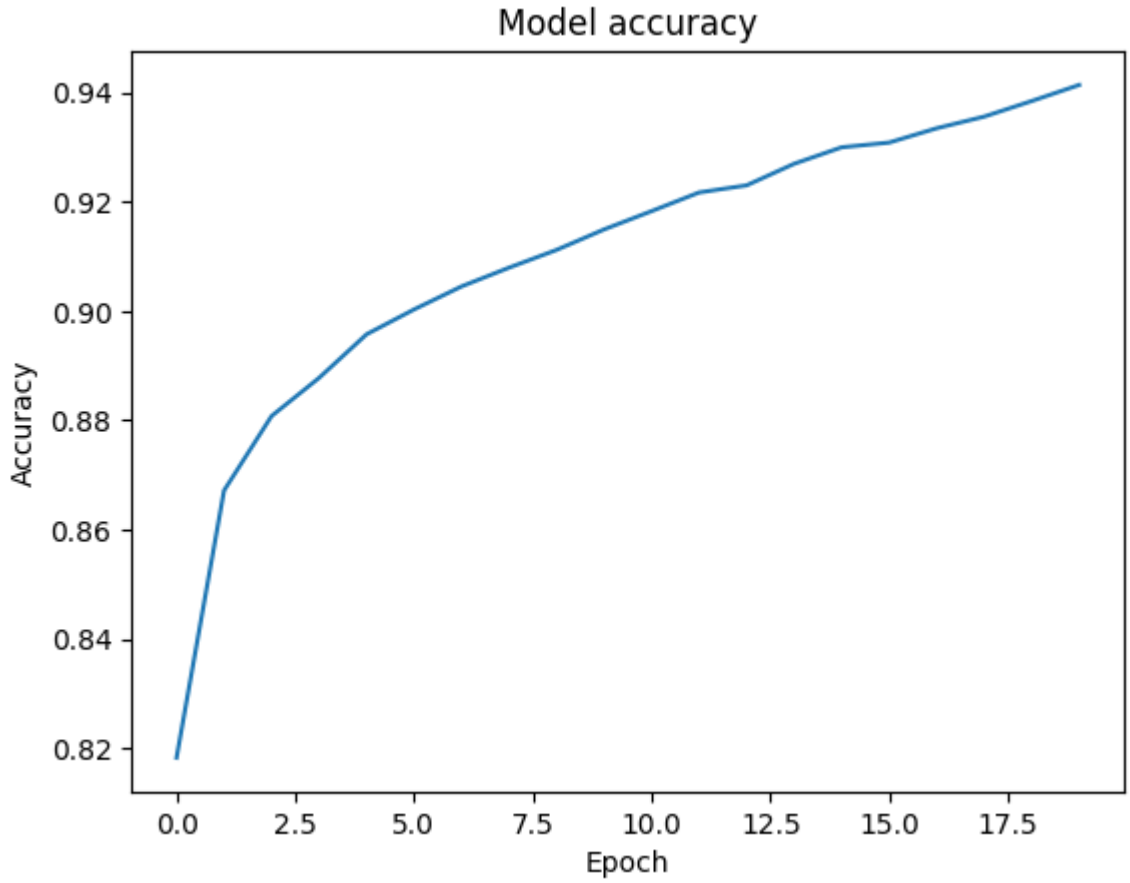
```
# Setup optimizer, loss function and metrics for the model
neuralNetwork.compile(
                optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy']
                )
```

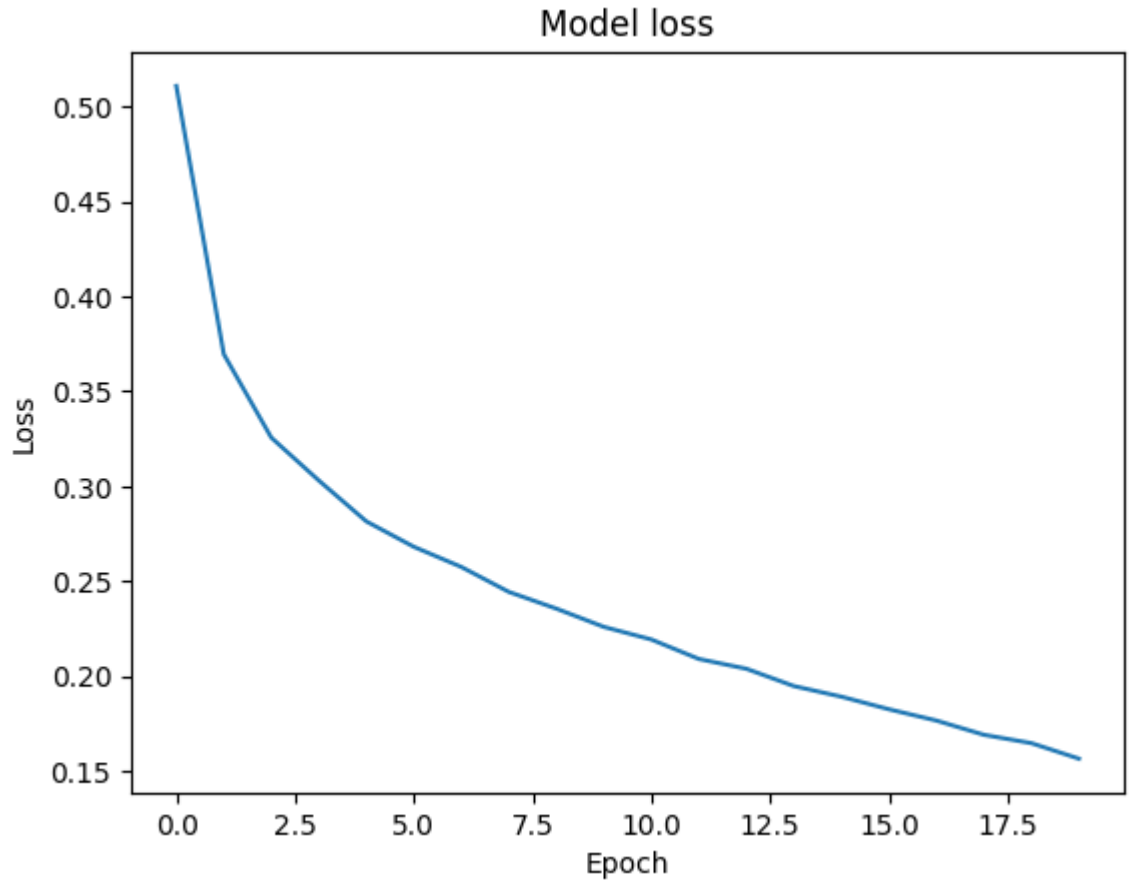
```
# Create model training log for TensorBoard
!rm -rf ./logs/

from datetime import datetime
import os

root_logdir = "logs"
run_id = datetime.now().strftime("%Y%m%d-%H%M%S")
logdir = os.path.join(root_logdir, run_id)

callbacks = [
    tf.keras.callbacks.TensorBoard(
        log_dir=logdir,
        histogram_freq=1
    )
]
```

```
# save the initial weights for later use
init_weights = neuralNetwork.get_weights()
```

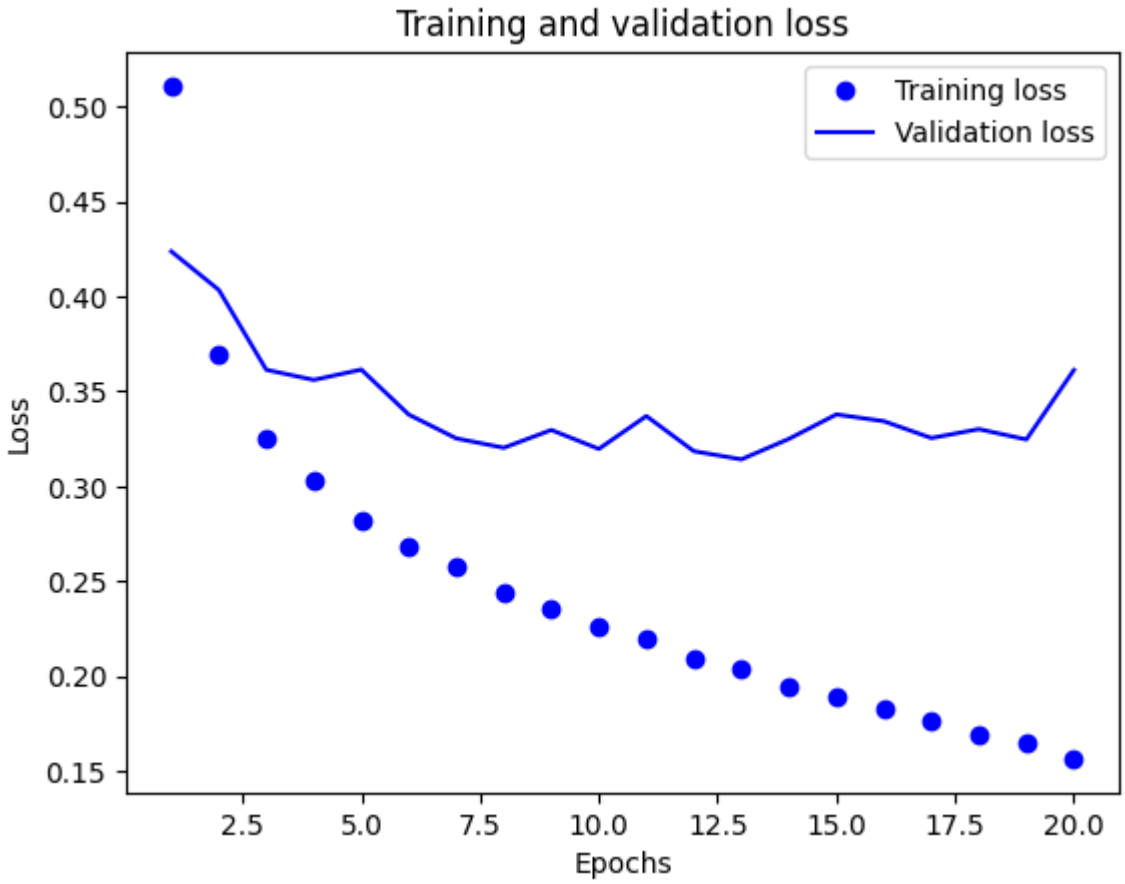
```
# Model fitting
history = neuralNetwork.fit(train_images,
                            train_labels,
                            epochs=20,
                            batch_size=128,
                            validation_data=(test_images, test_labels),
                            callbacks=callbacks)

# Evaluate the model by test dataset
test_loss, test_acc = neuralNetwork.evaluate(test_images, test_labels)
print(f'test_loss: {test_loss}')
print(f'test_acc: {test_acc}')
```

```
# Plot training accuracy values
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.show()

# Plot training loss values
plt.plot(history.history['loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()
```


Model accuracy
Accuracy
0.94
0.92
0.90
0.88
0.86
0.84
0.82
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch


Model loss
Loss
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch

```
# Check the training and validation loss
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

# "bo" is for "blue dot"
plt.plot(epochs, loss, 'bo', label='Training loss')
# b is for "solid blue line"
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()

results = neuralNetwork.evaluate(test_images, test_labels)
results
```




```
313/313 [==============================] - 1s 2ms/step - loss: 0.3613 - accura
[0.3612845242023468, 0.8896999955177307]
```

Analyse the model

```
# Print the model summary
neuralNetwork.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 256)               200960    

 dense_1 (Dense)             (None, 128)               32896     

 dense_2 (Dense)             (None, 10)                1290      

=================================================================
Total params: 235,146
Trainable params: 235,146
Non-trainable params: 0
_________________________________________________________________
```

```
# Count the number of parameters
num_params = neuralNetwork.count_params()
print('Number of parameters: ', num_params)
```

```
Number of parameters:  235146
```

```
# Print the number of activations
num_activations = 0
for layer in neuralNetwork.layers:
    if isinstance(layer, tf.keras.layers.Dense):
        num_activations += layer.output_shape[1]

print("Number of activations: ", num_activations)
```

```
Number of activations:  394
```

```
# Print the default initialization method of the dense layer

# create a new dense layer with default settings
dense_layer = Dense(units=32)

# get the kernel initialization method of the dense layer
init_method = dense_layer.kernel_initializer

print(init_method)
```

```
<keras.initializers.initializers.GlorotUniform object at 0x7fed783c9850>
```

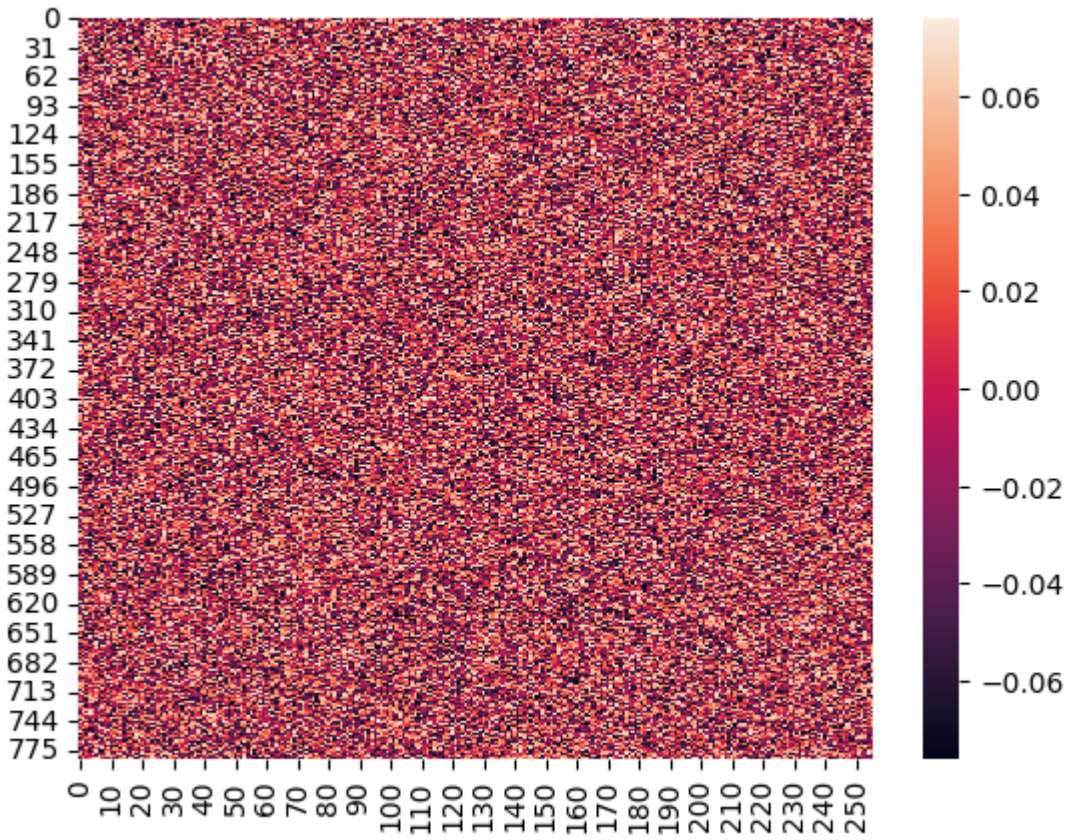
```
# Shape of the matrix of initial weights
init_weights[0].shape
```

```
(784, 256)
```

```

```

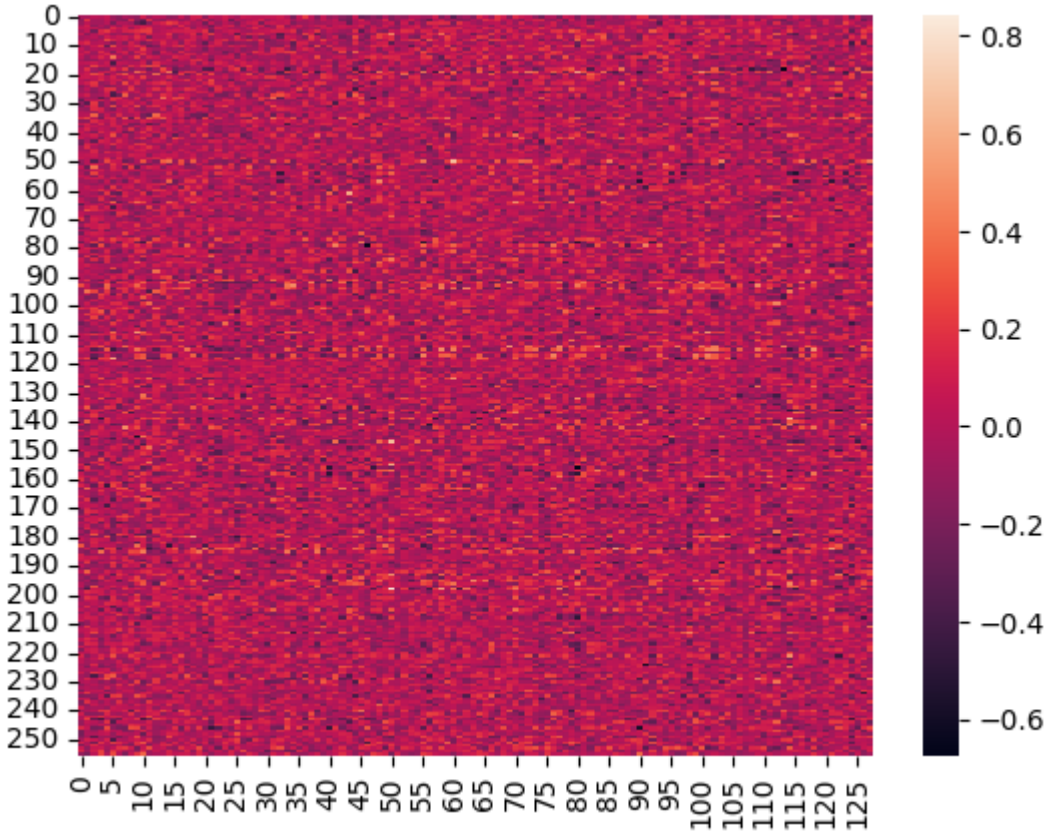
```
import seaborn as sns

# Get the initial weights of the model
matrix = init_weights[0]

# Plot the heatmap
sns.heatmap(matrix)
```

```
<Axes: >
```




```
# Get the weights of the second layer after model fitting
weights = neuralNetwork.layers[1].get_weights()
matrix = weights[0]

# Plot the heatmap
sns.heatmap(matrix)
```

```
<Axes: >
```

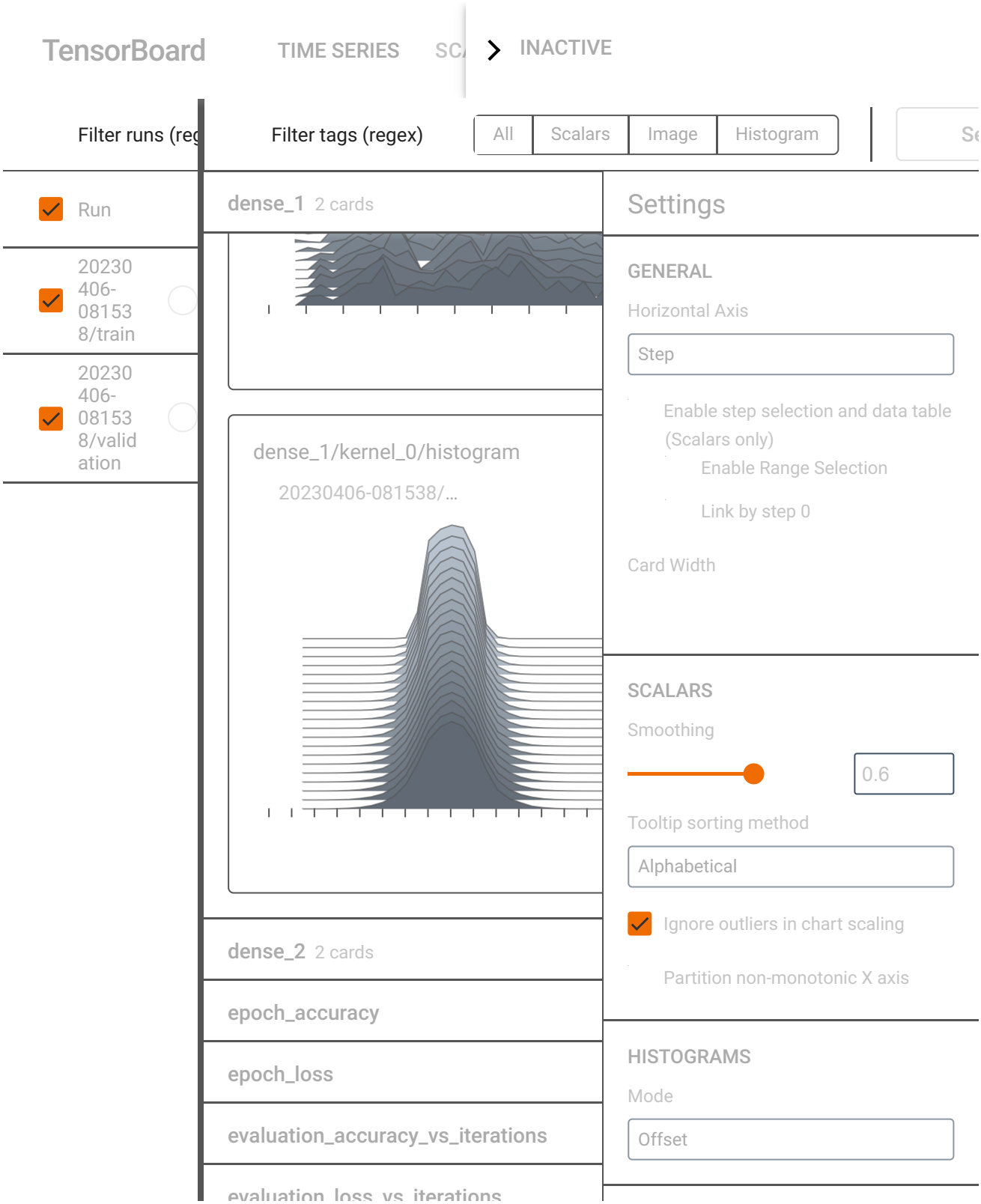



```
# Shape of the matrix of weights in second layer after model fitting
weights[0].shape
```

```
(256, 128)
```

Use TensorFlow tools

```
%reload_ext tensorboard
%tensorboard --logdir=logs
```



Improve the model by adding drop out layers

```
from keras.layers import Dense, Dropout

# Delete previous model
del neuralNetwork

# Recreate the model by applying dropout regularisation
neuralNetwork = models.Sequential([
    layers.Dense(256, activation='relu', input_shape=(28 * 28, )),
    layers.Dropout(0.5),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')])

# Setup optimizer, loss function and metrics for the model
neuralNetwork.compile(
                optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy']
                )

# Plot the model
plot_model(neuralNetwork, show_shapes=True)
```

```
# Clean up model training log for TensorBoard
!rm -rf ./logs/

# Model fitting
history = neuralNetwork.fit(train_images,
                            train_labels,
                            epochs=20,
                            batch_size=128,
                            validation_data=(test_images, test_labels),
                            callbacks=callbacks)
```

```
# Check the training and validation loss
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
```