```
!nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Tue_Aug_15_22:02:13_PDT_2023
Cuda compilation tools, release 12.2, V12.2.140
Build cuda_12.2.r12.2/compiler.33191640_0
```

```
!pip install git+https://github.com/andreinechaev/nvcc4jupyter.git
```

```
Collecting git+https://github.com/andreinechaev/nvcc4jupyter.git
  Cloning https://github.com/andreinechaev/nvcc4jupyter.git to /tmp/pip-req-build-ei625o8p
  Running command git clone --filter=blob:none --quiet https://github.com/andreinechaev/nvcc4jupyter.git /tmp/pip-req-build-ei625o8p
  Resolved https://github.com/andreinechaev/nvcc4jupyter.git to commit 5741c522547756ac4bb7a16df32106a15efb8a57
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
```

```
%load_ext nvcc4jupyter
```

```
Detected platform "Colab". Running its setup...
Source files will be saved in "/tmp/tmp6ar9c_mt".
```

```
%%cuda
#include<stdio.h>
#include<cuda.h>
#include<stdlib.h>
#include<time.h>

__global__ void max1(int* input)
{
 const int tid = threadIdx.x;

 auto step_size = 1;
 int number_of_threads = blockDim.x;
 int temp;

 while (number_of_threads > 0)
 {
   if (tid < number_of_threads) // still alive?
   {
     const auto fst = tid * step_size * 2;
     const auto snd = fst + step_size;
     //input[fst] += input[snd];
      if (input[fst]<input[snd])
      {
          temp=input[fst];
          input[fst]=input[snd];
          input[snd]=temp;
      }
   }
   __syncthreads();
   step_size <<= 1;
   number_of_threads >>= 1;
 }
}

int main()
{
 const auto count = 8;
 const int size = count * sizeof(int);
 int h[] = {13, 65, 15, 14, 33, 2, 30, 8};

 int* d;

 cudaMalloc(&d, size);
 cudaMemcpy(d, h, size, cudaMemcpyHostToDevice);

 max1 <<<1, count / 2 >>>(d);

 int result;
 cudaMemcpy(&result, d, sizeof(int), cudaMemcpyDeviceToHost);
   // cout << "Large no is %d " << result << endl;
printf("Large no is %d  ", result);

 getchar();

 cudaFree(d);
 //delete[] h;
```

```
  return 0;
}
```

```
Large no is 65
```

```
!nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Tue_Aug_15_22:02:13_PDT_2023
Cuda compilation tools, release 12.2, V12.2.140
Build cuda_12.2.r12.2/compiler.33191640_0
```

```
!pip install git+https://github.com/andreinechaev/nvcc4jupyter.git
```

```
Collecting git+https://github.com/andreinechaev/nvcc4jupyter.git
  Cloning https://github.com/andreinechaev/nvcc4jupyter.git to /tmp/pip-req-build-ei625o8p
  Running command git clone --filter=blob:none --quiet https://github.com/andreinechaev/nvcc4jupyter.git /tmp/pip-req-build-ei625o8p
  Resolved https://github.com/andreinechaev/nvcc4jupyter.git to commit 5741c522547756ac4bb7a16df32106a15efb8a57
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
```

```
%load_ext nvcc4jupyter
```

```
Detected platform "Colab". Running its setup...
Source files will be saved in "/tmp/tmp6ar9c_mt".
```

```
%%cuda
#include<stdio.h>
#include<cuda.h>
#include<stdlib.h>
#include<time.h>

__global__ void min1(int* input)
{
 const int tid = threadIdx.x;

 auto step_size = 1;
 int number_of_threads = blockDim.x;
 int temp;

 while (number_of_threads > 0)
 {
   if (tid < number_of_threads) // still alive?
   {
     const auto fst = tid * step_size * 2;
     const auto snd = fst + step_size;
     //input[fst] += input[snd];
      if (input[fst]>input[snd])
      {
          temp=input[fst];
          input[fst]=input[snd];
          input[snd]=temp;
      }
   }
   __syncthreads();
   step_size <<= 1;
   number_of_threads >>= 1;
 }
}

int main()
{
 const auto count = 8;
 const int size = count * sizeof(int);
 int h[] = {13, 65, 15, 14, 33, 23, 30, 8};

 int* d;

 cudaMalloc(&d, size);
 cudaMemcpy(d, h, size, cudaMemcpyHostToDevice);

 min1 <<<1, count / 2 >>>(d);

 int result;
 cudaMemcpy(&result, d, sizeof(int), cudaMemcpyDeviceToHost);
   // cout << "Large no is %d " << result << endl;
 printf("Small no is %d  ", result);

 getchar();

 cudaFree(d);
 //delete[] h;
```

```
 return 0;
}
```

```
Small no is 8
```

```
!nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Tue_Aug_15_22:02:13_PDT_2023
Cuda compilation tools, release 12.2, V12.2.140
Build cuda_12.2.r12.2/compiler.33191640_0
```

```
!pip install git+https://github.com/andreinechaev/nvcc4jupyter.git
```

```
Collecting git+https://github.com/andreinechaev/nvcc4jupyter.git
  Cloning https://github.com/andreinechaev/nvcc4jupyter.git to /tmp/pip-req-build-ei625o8p
  Running command git clone --filter=blob:none --quiet https://github.com/andreinechaev/nvcc4jupyter.git /tmp/pip-req-build-ei625o8p
  Resolved https://github.com/andreinechaev/nvcc4jupyter.git to commit 5741c522547756ac4bb7a16df32106a15efb8a57
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
```

```
%load_ext nvcc4jupyter
```

```
Detected platform "Colab". Running its setup...
Source files will be saved in "/tmp/tmp6ar9c_mt".
```

```
%%cuda
#include<stdio.h>
#include<cuda.h>
#include<stdlib.h>
#include<time.h>

__global__ void sum(int* input)
{
 const int tid = threadIdx.x;

 auto step_size = 1;
 int number_of_threads = blockDim.x;

 while (number_of_threads > 0)
 {
   if (tid < number_of_threads) // still alive?
   {
     const auto fst = tid * step_size * 2;
 const auto snd = fst + step_size;
     input[fst] += input[snd];
   }
   __syncthreads();
   step_size <<= 1;
number_of_threads >>= 1;
}
 }

 int main()
 {
  const auto count = 8;
  const int size = count * sizeof(int);
int h[] = {13, 27, 15, 14, 33, 2, 30, 8};

  int* d;

  cudaMalloc(&d, size);
  cudaMemcpy(d, h, size, cudaMemcpyHostToDevice);

  sum <<<1, count / 2 >>>(d);

  int result;
  cudaMemcpy(&result, d, sizeof(int), cudaMemcpyDeviceToHost);
    // cout << "Sum is " << result << endl;
  printf("Sum is %d  ", result);

  getchar();

  cudaFree(d);
  //delete[] h;

  return 0;
 }
```

```
Sum is 142
```

```
!nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Tue_Aug_15_22:02:13_PDT_2023
Cuda compilation tools, release 12.2, V12.2.140
Build cuda_12.2.r12.2/compiler.33191640_0
```

```
!pip install git+https://github.com/andreinechaev/nvcc4jupyter.git
```

```
Collecting git+https://github.com/andreinechaev/nvcc4jupyter.git
  Cloning https://github.com/andreinechaev/nvcc4jupyter.git to /tmp/pip-req-build-438dnets
  Running command git clone --filter=blob:none --quiet https://github.com/andreinechaev/nvcc4jupyter.git /tmp/pip-req-build-438dnets
  Resolved https://github.com/andreinechaev/nvcc4jupyter.git to commit 5741c522547756ac4bb7a16df32106a15efb8a57
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Building wheels for collected packages: nvcc4jupyter
  Building wheel for nvcc4jupyter (pyproject.toml) ... done
  Created wheel for nvcc4jupyter: filename=nvcc4jupyter-1.2.1-py3-none-any.whl size=10739 sha256=652cfa192a926bb82952bbed8fb3fd55f14
  Stored in directory: /tmp/pip-ephem-wheel-cache-2wx_nqnt/wheels/a8/b9/18/23f8ef71ceb0f63297dd1903aedd067e6243a68ea756d6feea
Successfully built nvcc4jupyter
Installing collected packages: nvcc4jupyter
Successfully installed nvcc4jupyter-1.2.1
```

```
%load_ext nvcc4jupyter
```

```
Detected platform "Colab". Running its setup...
Source files will be saved in "/tmp/tmp6jg70rkt".
```

```
%%cuda
/*[m,n]*[n,j]= [m,j]
first take transpose of vector anfd
then mulriply and add row wisw for each col.
*/
//parallelism of multiplication only
//tested ok on 05/12/2018

#include <cuda.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define m 10
__global__ void mul_r(int *a, int *b, int *c){

     int tid = threadIdx.x;
     if (tid < m){
           c[tid]= a[tid] * b[tid];
           }

   }



int main(){
     int  n, c, d, fst[10][10], snd[10][10], t_snd[10][10];
     int row,col,sum_c, a[10], b[10], ans[10];

 /*
   printf("Enter the number of rows and columns of matrix\n");
   scanf("%d%d", &m, &n);
   */
n=m;  //square matrix only
   //printf("Enter the elements of first matrix\n");
   //for true randum value of vector
//srand(time(0));
   for (c = 0; c < m; c++)
   {

      for (d = 0; d < n; d++)
   {
        //scanf("%d", &first[c][d]);

    fst[c][d]=rand()%10+1;
   }
 }
 printf("display the elements of first matrix\n");
 for (c = 0; c < m; c++) {
     for (d = 0 ; d < n; d++) {
```

```
            printf("%d\t", fst[c][d]);
              }
               printf("\n");
            }

  // take next matrix
  //for true randum value of vector
  //srand(time(0));
    for (c = 0; c < m; c++)
    {

       for (d = 0; d < n; d++)
    {
          //scanf("%d", &first[c][d]);

     snd[c][d]=rand()%10+1;
    }
  }
  printf("display the elements of second matrix\n");
  for (c = 0; c < m; c++) {
       for (d = 0 ; d < n; d++) {

            printf("%d\t", snd[c][d]);
              }
               printf("\n");
            }
  // transpose of  second matrix
  for(c=0; c<m; c++)
         for(d=0; d<n; d++)
         {
             t_snd[d][c] = snd[c][d];
         }

     // Displaying the transpose of matrix a
     printf("\nTranspose of second Matrix:\n");
     for (c = 0; c < n; c++) {
       for (d = 0 ; d < m; d++) {

           printf("%d\t", t_snd[c][d]);

               }
               printf("\n");
            }

     // now multiply on cuda
  int *dev_a, *dev_b,*dev_ans;
  cudaError_t err=cudaSuccess;
  // allocate memory on GPU
  err=cudaMalloc((void**)&dev_a,m * sizeof(int));
  if (err !=cudaSuccess)
  {   printf("failed to  allocate on device \n");
  printf("error is:\n",cudaGetErrorString(err));
  exit(EXIT_FAILURE);
  }
  //printf("first  ok\n");
  cudaMalloc((void**)&dev_b,m * sizeof(int));
  cudaMalloc((void**)&dev_ans,m * sizeof(int));
  //printf("first finished ok\n");
  row=0;
  col=0;
   cudaEvent_t start, end;
   cudaEventCreate(&start);
   cudaEventCreate(&end);
   cudaEventRecord(start);

  for(row=0; row<m; row++){

   for (d = 0 ; d < m; d++) {

           a[d]=fst[row][d];

               }
              // printf("ok a\n");
    cudaMemcpy(dev_a,a,m*sizeof(int), cudaMemcpyHostToDevice);
  for (col=0; col<m; col++){
   for (d= 0 ; d < m; d++) {

           b[d]=t_snd[col][d];
           ans[d]=0;

               }
              // printf("ok b\n");
    cudaMemcpy(dev_b,b,m*sizeof(int), cudaMemcpyHostToDevice);
```

```
  cudaMemcpy(dev_ans,ans,m*sizeof(int), cudaMemcpyHostToDevice);
//  printf("calling GPU\n");
  mul_r<<<1,m>>>(dev_a,dev_b,dev_ans);
  err=cudaMemcpy(ans,dev_ans,m*sizeof(int), cudaMemcpyDeviceToHost);
  if (err !=cudaSuccess)
    {   printf("failed to  copy from device \n");
      exit(EXIT_FAILURE);
    }
//printf("GPU returned\n");
//a=fst[0];
sum_c=0;
 for (d = 0 ; d < m; d++) {

          //printf("%d\t", ans[d]);
        sum_c+=ans[d];
             }
snd[row][col]=sum_c;



//printf("one element=%d\n",snd[row][col]);
             // printf("\n");

}
}
//
cudaEventRecord(end);
 cudaEventSynchronize(end);
 float time = 0;
 cudaEventElapsedTime(&time, start, end);
 printf("execution time=%f\n",time);
//
 printf(" Matrix multipliation ans=:\n");
   for (c = 0; c < n; c++) {
     for (d = 0 ; d < m; d++) {

          printf("%d\t", snd[c][d]);

              }
               printf("\n");
            }
//
    return 0;
      }
```

```
   display the elements of first matrix
   4       7       8       6       4       6       7       3       10      2
   3       8       1       10      4       7       1       7       3       7
   2       9       8       10      3       1       3       4       8       6
   10      3       3       9       10      8       4       7       2       3
   10      4       2       10      5       8       9       5       6       1
   4       7       2       1       7       4       3       1       7       2
   6       6       5       8       7       6       7       10      4       8
   5       6       3       6       5       8       5       5       4       1
   8       9       7       9       9       5       4       2       5       10
   3       1       7       9       10      3       7       7       5       10
   display the elements of second matrix
   6       1       5       9       8       2       8       3       8       3
   3       7       2       1       7       2       6       10      5       10
   1       10      2       8       8       2       2       6       10      8
   8       7       8       4       7       6       7       4       10      5
   9       2       3       10      4       10      1       9       9       6
   1       10      7       4       9       6       7       2       2       6
   10      9       5       9       2       1       4       1       5       5
   5       5       8       7       4       2       8       6       10      7
   3       2       8       9       6       8       5       2       9       6
   10      8       6       4       9       9       4       2       9       10

   Transpose of second Matrix:
   6       3       1       8       9       1       10      5       3       10
   1       7       10      7       2       10      9       5       2       8
   5       2       2       8       3       7       5       8       8       6
   9       1       8       4       10      4       9       7       9       4
   8       7       8       7       4       9       2       4       6       9
   2       2       2       6       10      6       1       2       8       9
   8       6       2       7       1       7       4       8       5       4
   3       10      6       4       9       2       1       6       2       2
   8       5       10      10      9       2       5       10      9       9
   3       10      8       5       6       6       5       7       6       10
   execution time=3.148256
    Matrix multipliation ans=:
   278     357     303     377     361     261     288     251     428     372
   290     323     301     264     348     268     300     248     389     355
   289     342     287     316     358     263     274     268     451     385
   353     323     330     400     375     295     327     276     456     348
   350     344     352     406     369     266     349     236     439     340
```

```
198  196  186  254  238  205  183  196  274  253
404  413  374  427  418  319  359  306  526  445
249  295  265  301  303  218  269  223  341  301
405  408  342  418  463  360  336  329  535  463
413  381  345  429  379  345  287  272  525  412
```

Start coding or generate with AI.

```
!nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Tue_Aug_15_22:02:13_PDT_2023
Cuda compilation tools, release 12.2, V12.2.140
Build cuda_12.2.r12.2/compiler.33191640_0
```

```
!pip install git+https://github.com/andreinechaev/nvcc4jupyter.git
```

```
Collecting git+https://github.com/andreinechaev/nvcc4jupyter.git
  Cloning https://github.com/andreinechaev/nvcc4jupyter.git to /tmp/pip-req-build-438dnets
  Running command git clone --filter=blob:none --quiet https://github.com/andreinechaev/nvcc4jupyter.git /tmp/pip-req-build-438dnets
  Resolved https://github.com/andreinechaev/nvcc4jupyter.git to commit 5741c522547756ac4bb7a16df32106a15efb8a57
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Building wheels for collected packages: nvcc4jupyter
  Building wheel for nvcc4jupyter (pyproject.toml) ... done
  Created wheel for nvcc4jupyter: filename=nvcc4jupyter-1.2.1-py3-none-any.whl size=10739 sha256=652cfa192a926bb82952bbed8fb3fd55f14
  Stored in directory: /tmp/pip-ephem-wheel-cache-2wx_nqnt/wheels/a8/b9/18/23f8ef71ceb0f63297dd1903aedd067e6243a68ea756d6feea
Successfully built nvcc4jupyter
Installing collected packages: nvcc4jupyter
Successfully installed nvcc4jupyter-1.2.1
```

```
%load_ext nvcc4jupyter
```

```
Detected platform "Colab". Running its setup...
Source files will be saved in "/tmp/tmp6jg70rkt".
```

```
%%cuda
#include<stdio.h>
#include<cuda.h>
#include<stdlib.h>
#include<time.h>


#define N 500

__global__ void add(int *a, int *b, int *c){

int tid = threadIdx.x;
if (tid < N){
 c[tid]= a[tid] + b[tid];
 }

}

 int main (void){
int a[N], b[N], c[N];
int *dev_a, *dev_b,*dev_c;
cudaError_t err=cudaSuccess;
err=cudaMalloc((void**)&dev_a,N * sizeof(int));
if (err !=cudaSuccess)
{  printf("failed to  allocate on device \n");
 printf("error is:\n",cudaGetErrorString(err));
exit(EXIT_FAILURE);
}
cudaMalloc((void**)&dev_b,N * sizeof(int));
cudaMalloc((void**)&dev_c,N * sizeof(int));

for(int i=0;i<N;i++){
a[i] =i;
b[i] = i*i;

c[i]=0;
}

/*for(int i=0;i<N;i++){
printf(" c contents are =%d\n",  c[i]);
} */
cudaEvent_t start, end;
cudaEventCreate(&start);
cudaEventCreate(&end);
cudaEventRecord(start);

cudaMemcpy(dev_a,a,N*sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(dev_b,b,N*sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(dev_c,c,N*sizeof(int), cudaMemcpyHostToDevice);
add<<<1,N>>>(dev_a,dev_b,dev_c);
```

```
 err=cudaMemcpy(c,dev_c,N*sizeof(int), cudaMemcpyDeviceToHost);
 if (err !=cudaSuccess)
 {  printf("failed to  copy from device \n");
 exit(EXIT_FAILURE);
 }

 cudaEventRecord(end);
 cudaEventSynchronize(end);
 float time = 0;
cudaEventElapsedTime(&time, start, end);
printf("execution time=%f\n",time);

 for(int i=0;i<N;i++){
 //printf("%d +%d=%d\n", a[i], b[i], c[i]);
 }

 cudaFree(dev_a);
cudaFree(dev_b);
 cudaFree(dev_c);

 return 0;
 }
```

```
execution time=0.202368
```

Assignment 3

In [3]:
```
from __future__ import absolute_import, division, print_function
# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras
# Helper libraries
import numpy as np
import matplotlib.pyplot as plt
```

In [4]:
```
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.loa
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-da
tasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-da
tasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 6s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-da
tasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-da
tasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 1s 0us/step
```

In [5]:
```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

In [6]:
```
train_images.shape
```

Out[6]:
```
(60000, 28, 28)
```

In [7]:
```
len(train_labels)
```

Out[7]:
```
60000
```

In [8]:
```
train_labels
```

Out[8]:
```
array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

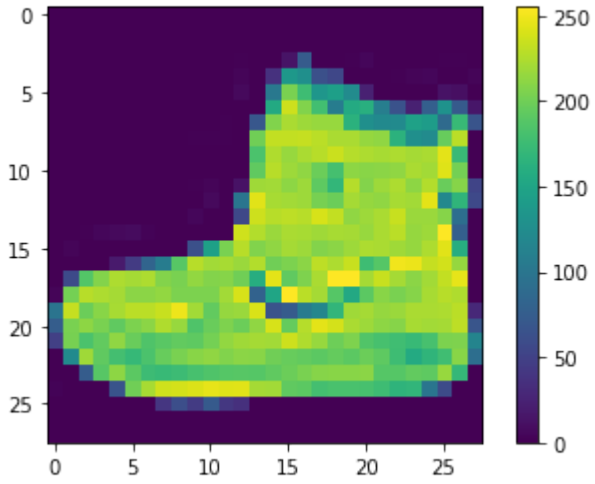
In [9]:
```
test_images.shape
```

Out[9]:
```
(10000, 28, 28)
```

In [10]:
```
len(test_labels)
```

Out[10]:
```
10000
```

In [11]:
```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```



In [12]:
```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

In [16]:
```
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()
```

In [17]:
```
model = keras.Sequential([
keras.layers.Flatten(input_shape=(28, 28)),
keras.layers.Dense(128, activation=tf.nn.relu),
keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

In [18]:
```
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
```

In [19]:
```
model.fit(train_images, train_labels, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 2s 811us/step - loss: 0.4927 -
accuracy: 0.8278
Epoch 2/5
1875/1875 [==============================] - 1s 758us/step - loss: 0.3689 -
accuracy: 0.8667
Epoch 3/5
1875/1875 [==============================] - 2s 828us/step - loss: 0.3341 -
accuracy: 0.8782
Epoch 4/5
1875/1875 [==============================] - 1s 747us/step - loss: 0.3102 -
accuracy: 0.8862
Epoch 5/5
1875/1875 [==============================] - 2s 811us/step - loss: 0.2929 -
accuracy: 0.8919
```

Out[19]:
```
<keras.src.callbacks.History at 0x17f7cfaf0>
```

In [20]:
```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

```
313/313 [==============================] - 0s 552us/step - loss: 0.3479 - a
ccuracy: 0.8757
Test accuracy: 0.8756999969482422
```

In [21]:
```
predictions = model.predict(test_images)
```

```
313/313 [==============================] - 0s 468us/step
```

In [22]:
```
predictions[0]
```

Out[22]:
```
array([6.44061117e-07, 8.45806625e-09, 9.75436592e-07, 5.93774641e-10,
       8.83455755e-07, 1.17749525e-02, 1.09047915e-06, 6.85961824e-03,
       1.04943738e-05, 9.81351376e-01], dtype=float32)
```

In [23]:
```
np.argmax(predictions[0])
```

Out[23]:
```
9
```

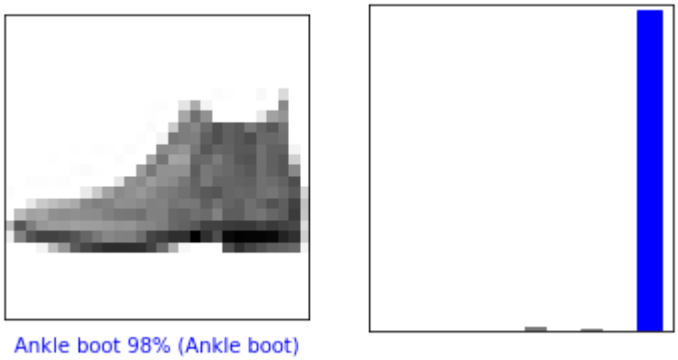
In [24]:
```
test_labels[0]
```

Out[24]:
```
9
```

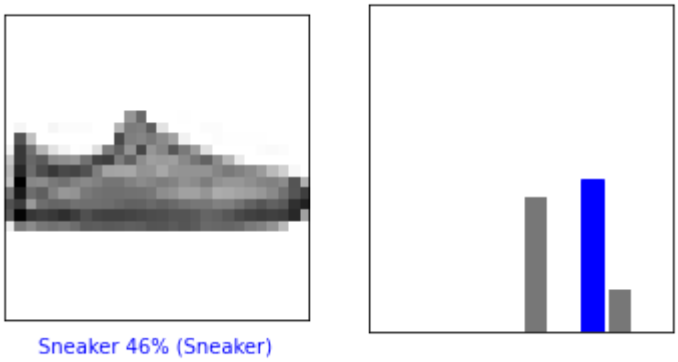
In [25]:
```
def plot_image(i, predictions_array, true_label, img):
    predictions_array, true_label, img = predictions_array[i], true_label[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    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):
    predictions_array, true_label = predictions_array[i], true_label[i]
    plt.grid(False)
    plt.xticks([])
    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')
```

In [26]:
```
i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1,2,2)
```

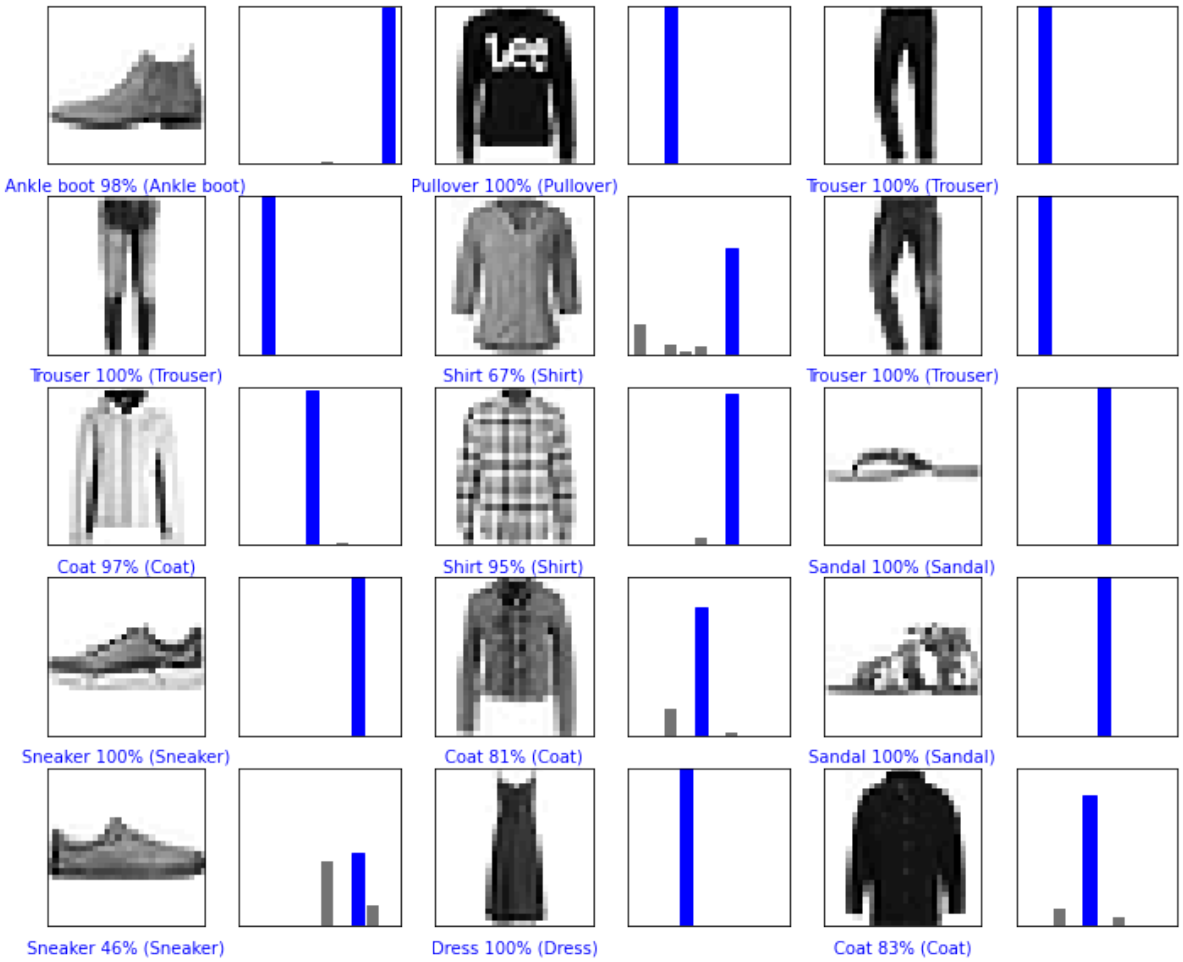
```
plot_value_array(i, predictions, test_labels)
plt.show()
```



In [27]:
```
i = 12
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions, test_labels)
plt.show()
```



In [28]:
```
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, test_labels, test_images)
    plt.subplot(num_rows, 2*num_cols, 2*i+2)
    plot_value_array(i, predictions, test_labels)
plt.show()
```



In [29]:
```
img = test_images[0]
print(img.shape)
```

```
(28, 28)
```

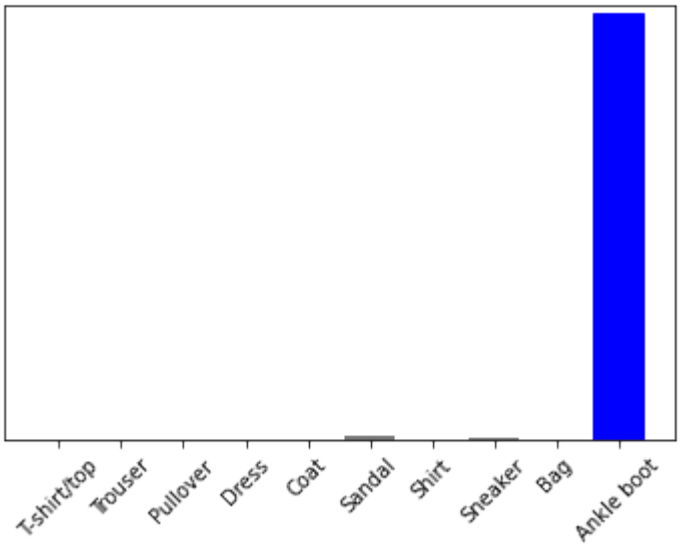
In [30]:
```
img = (np.expand_dims(img,0))
print(img.shape)
```

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

In [31]:
```
predictions_single = model.predict(img)
print(predictions_single)
```

```
1/1 [==============================] - 0s 13ms/step
[[6.4406112e-07 8.4580494e-09 9.7543830e-07 5.9377464e-10 8.8345740e-07
  1.1774947e-02 1.0904781e-06 6.8596182e-03 1.0494354e-05 9.8135138e-01]]
```

In [32]:
```
plot_value_array(0, predictions_single, test_labels)
_ = plt.xticks(range(10), class_names, rotation=45)
```



In [33]:
```
np.argmax(predictions_single[0])
```

Out[33]:
```
9
```