ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

Classification de base : classer les images de vêtements

Ce tp va vous permettre d'appliquer un modèle de réseau neuronal pour classer les images de vêtements, comme les baskets et les chemises. Il s'agit d'un aperçu rapide d'un programme TensorFlow.

On va utiliser tf.keras , une API de haut niveau pour créer et former des modèles dans TensorFlow.

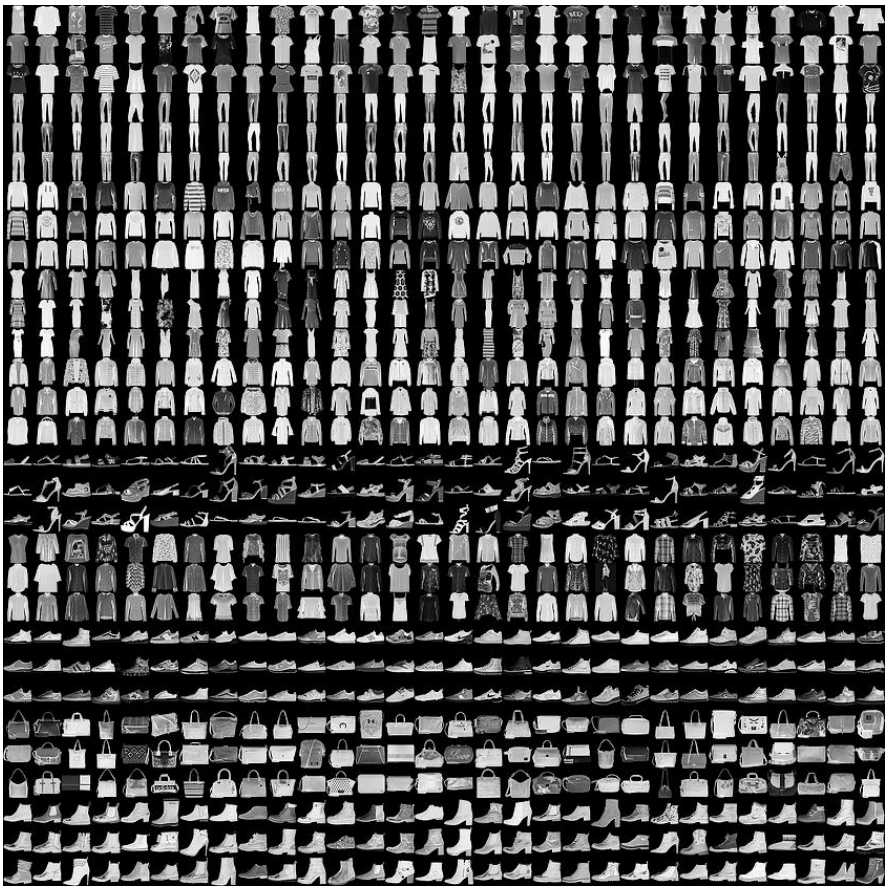
```
import tensorflow as tf
```

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

Importer le jeu de données Fashion MNIST

On va utiliser le jeu de données Fashion MNIST qui contient 70 000 images en niveaux de gris dans 10 catégories. Les images montrent des vêtements individuels à basse résolution (28 par 28 pixels), comme on le voit ici :



Le jeu de données MNIST contient des images de chiffres manuscrits (0, 1, 2, etc.) dans un format identique à celui des vêtements que vous utiliserez ici.

Ici, 60 000 images sont utilisées pour former le réseau et 10 000 images pour évaluer la précision avec laquelle le réseau a appris à classer les images. Vous pouvez accéder au Fashion MNIST directement depuis TensorFlow. Importez et chargez les données Fashion MNIST directement depuis TensorFlow :

```
fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 4s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0s/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 1s 0us/step
```

Le chargement de l'ensemble de données renvoie quatre tableaux NumPy :

Les tableaux train_images et train_labels sont l' ensemble d'apprentissage - les données que le modèle utilise pour apprendre.

Le modèle est testé par rapport à l' ensemble de tests , aux tableaux test_images et test_labels .

Les images sont des tableaux NumPy 28x28, avec des valeurs de pixels allant de 0 à 255. Les étiquettes sont un tableau d'entiers, allant de 0 à 9. Ceux-ci correspondent à la classe de vêtements que l'image représente :

Étiqueter	Classer
0	T-shirt/haut
1	Pantalon
2	Arrêtez-vous
3	Robe
4	Manteau
5	Sandale
6	La chemise
sept	Basket
8	Sac
9	Bottine

Chaque image est associée à une seule étiquette. Étant donné que les noms de classe ne sont pas inclus dans le jeu de données, stockez-les ici pour les utiliser ultérieurement lors du traçage des images :

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

Explorer les données

Explorons le format de l'ensemble de données avant de former le modèle. L'exemple suivant montre qu'il y a 60 000 images dans l'ensemble d'apprentissage, chaque image étant représentée en 28 x 28 pixels :

```
train_images.shape
```

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

De même, il y a 60 000 libellés dans l'ensemble d'apprentissage :

```
e [8]: len(train_labels)
ut[8]: 60000
```

Chaque étiquette est un entier compris entre 0 et 9 :

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

Il y a 10 000 images dans le jeu de test. Encore une fois, chaque image est représentée en 28 x 28 pixels :

```
 [10]: test_images.shape
t[10]: (10000, 28, 28)
```

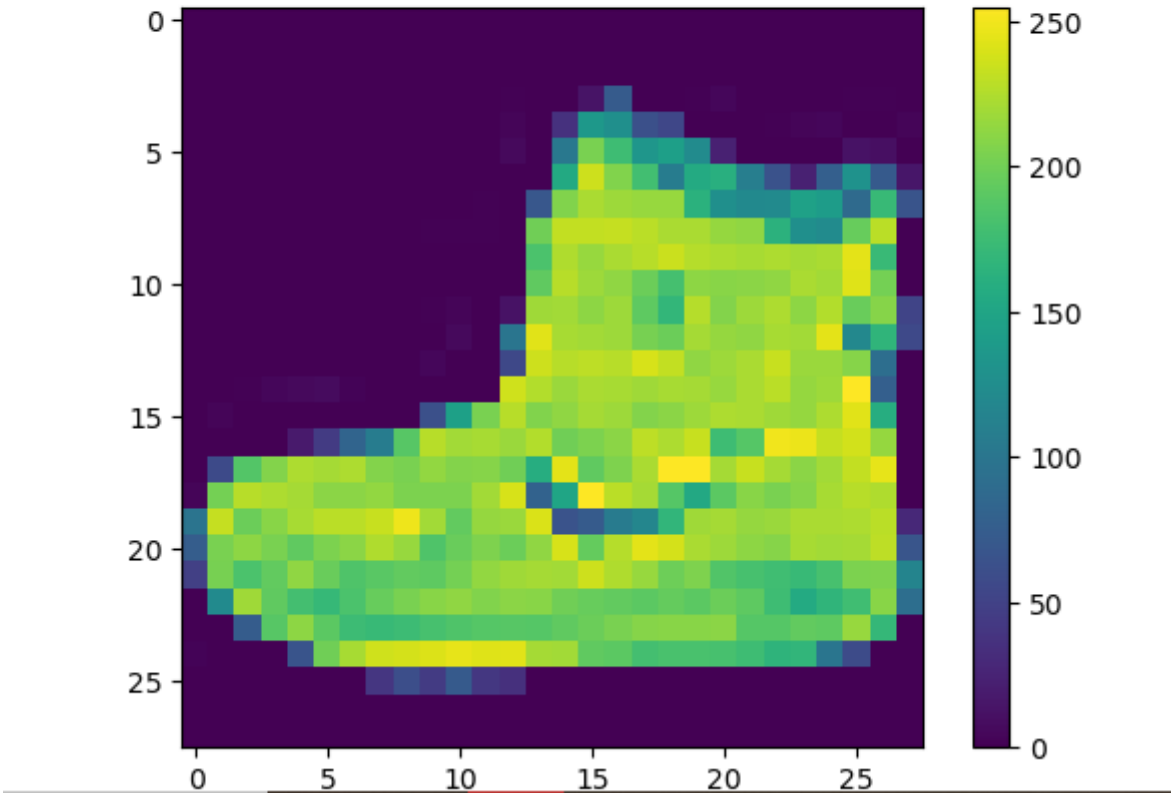
Et le jeu de test contient 10 000 étiquettes d'images :

```
 [11]: len(test_labels)
t[11]: 10000
```

Prétraiter les données

Les données doivent être prétraitées avant de former le réseau. Si vous inspectez la première image de l'ensemble d'apprentissage, vous verrez que les valeurs de pixel se situent dans la plage de 0 à 255 :

[12]:
```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```



Mettez ces valeurs à l'échelle dans une plage de 0 à 1 avant de les alimenter au modèle de réseau neuronal. Pour ce faire, divisez les valeurs par 255. Il est important que l'ensemble d'apprentissage et l' ensemble de test soient prétraités de la même manière :

[13]:
```
train_images = train_images / 255.0

test_images = test_images / 255.0
```

Pour vérifier que les données sont au format correct et que vous êtes prêt à construire et former le réseau, affichons les 25 premières images de l'ensemble de formation et affichons le nom de la classe sous chaque image.

[14]:
```
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()
```

Construire le modèle

Construire le réseau de neurones nécessite de configurer les couches du modèle, puis de compiler le modèle.

Mettre en place les calques

Le bloc de construction de base d'un réseau de neurones est la couche. Les couches extraient des représentations des données qui y sont introduites. Espérons que ces représentations sont significatives pour le problème à résoudre.

L'essentiel de l'apprentissage en profondeur consiste à enchaîner des couches simples.

[15]:
```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])
```

La première couche de ce réseau, tf.keras.layers.Flatten , transforme le format des images d'un tableau bidimensionnel (de 28 par 28 pixels) en un tableau unidimensionnel (de 28 * 28 = 784 pixels). Considérez ce calque comme désempilant des rangées de pixels dans l'image et en les alignant. Cette couche n'a aucun paramètre à apprendre ; il ne fait que reformater les données.

Une fois les pixels aplatis, le réseau se compose d'une séquence de deux couches tf.keras.layers.Dense . Ce sont des couches neuronales densément connectées ou entièrement connectées. La première couche Dense compte 128 nœuds (ou neurones). La deuxième (et dernière) couche renvoie un tableau logits d'une longueur de 10. Chaque nœud contient un score indiquant que l'image actuelle appartient à l'une des 10 classes.

Compiler le modèle

Avant que le modèle ne soit prêt pour l'entraînement, il a besoin de quelques réglages supplémentaires. Ceux-ci sont ajoutés lors de l'étape de compilation du modèle :

Fonction de perte : mesure la précision du modèle pendant l'entraînement. Vous souhaitez minimiser cette fonction pour "orienter" le modèle dans la bonne direction.

Optimiseur : c'est ainsi que le modèle est mis à jour en fonction des données qu'il voit et de sa fonction de perte.

Métriques : utilisées pour surveiller les étapes de formation et de test. L'exemple suivant utilise precision, la fraction des images qui sont correctement classées.

```
[16]: model.compile(optimizer='adam',
                    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                    metrics=['accuracy'])
```

Former le modèle

L'entraînement du modèle de réseau neuronal nécessite les étapes suivantes :

Transférez les données de formation au modèle. Dans cet exemple, les données d'apprentissage se trouvent dans les tableaux train_images et train_labels.

Le modèle apprend à associer des images et des étiquettes.

Vous demandez au modèle de faire des prédictions sur un jeu de test — dans cet exemple, le tableau test_images .

Vérifiez que les prédictions correspondent aux étiquettes du tableau test_labels .

Nourrir le modèle

Pour commencer l'entraînement, appelez la méthode model.fit , ainsi appelée car elle "adapte" le modèle aux données d'entraînement :

```
[17]: model.fit(train_images, train_labels, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.5004 - accuracy: 0.8241
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3747 - accuracy: 0.8649
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3379 - accuracy: 0.8768
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3127 - accuracy: 0.8849
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2947 - accuracy: 0.8916
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2803 - accuracy: 0.8959
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2682 - accuracy: 0.9010
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2563 - accuracy: 0.9041
Epoch 9/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2476 - accuracy: 0.9082
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2379 - accuracy: 0.9098
```


Au fur et à mesure que le modèle s'entraîne, les métriques de perte et de précision sont affichées. Ce modèle atteint une précision d'environ 0,91 (ou 91%) sur les données d'apprentissage.

Évaluer la précision

Ensuite, comparez les performances du modèle sur l'ensemble de données de test :

```
[18]: test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

      print('\nTest accuracy:', test_acc)

      313/313 - 1s - loss: 0.3469 - accuracy: 0.8823 - 621ms/epoch - 2ms/step

      Test accuracy: 0.8823000192642212
```

Il s'avère que la précision sur l'ensemble de données de test est un peu inférieure à la précision sur l'ensemble de données d'apprentissage. Cet écart entre la précision de l'entraînement et la précision des tests représente un surapprentissage. Le surajustement se produit lorsqu'un modèle d'apprentissage automatique fonctionne moins bien sur de nouvelles entrées inédites que sur les données d'apprentissage. Un modèle surajusté "mémorise" le bruit et les détails dans l'ensemble de données d'apprentissage à un point tel qu'il a un impact négatif sur les performances du modèle sur les nouvelles données.

Faire des prédictions

Avec le modèle formé, vous pouvez l'utiliser pour faire des prédictions sur certaines images. Attachez une couche softmax pour convertir les sorties linéaires du modèle (logits) en probabilités, ce qui devrait être plus facile à interpréter.

```
[19]: probability_model = tf.keras.Sequential([model,
                                               tf.keras.layers.Softmax()])
```

Ici, le modèle a prédit l'étiquette pour chaque image de l'ensemble de test. Reprenons la première prédiction :

```
[20]: predictions = probability_model.predict(test_images)

      313/313 [==============================] - 1s 1ms/step
```

```
[21]: predictions[0]

t[21]: array([4.1277048e-08, 2.4261024e-09, 3.0521516e-09, 1.9756228e-09,
              6.4854571e-09, 1.2011963e-03, 5.4415672e-09, 1.1751978e-03,
              1.0367216e-07, 9.9762338e-01], dtype=float32)
```

Une prédiction est un tableau de 10 nombres. Ils représentent la "confiance" du modèle que l'image correspond à chacun des 10 vêtements différents. Vous pouvez voir quelle étiquette a la valeur de confiance la plus élevée :

```
[22]: np.argmax(predictions[0])

t[22]: 9
```

Ainsi, le modèle est plus sûr que cette image est une bottine, ou class_names[9] . L'examen de l'étiquette du test montre que cette classification est correcte :

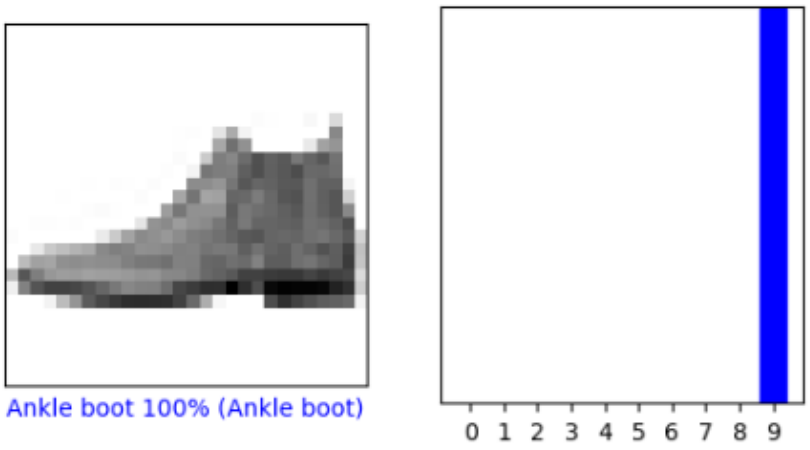
```
[23]: test_labels[0]
```

```
t[23]: 9
```

Représentez-le graphiquement pour examiner l'ensemble complet des 10 prédictions de classe.

```
[24]: 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')
```

```
[25]: 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()
```

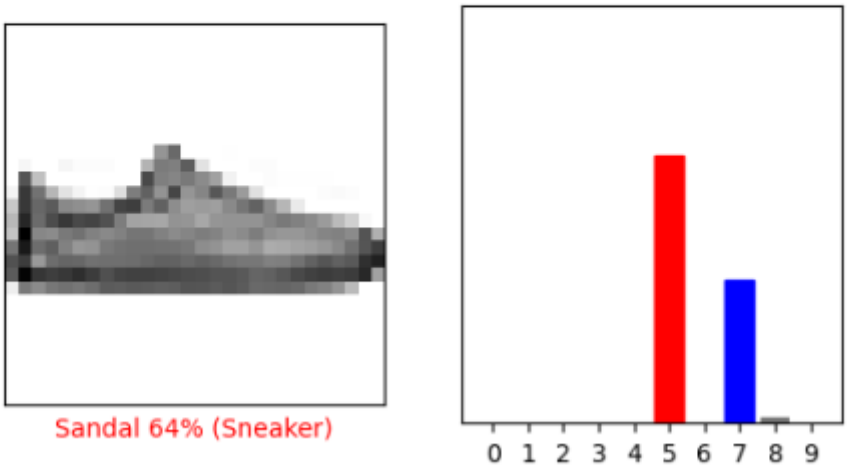


Vérifier les prédictions

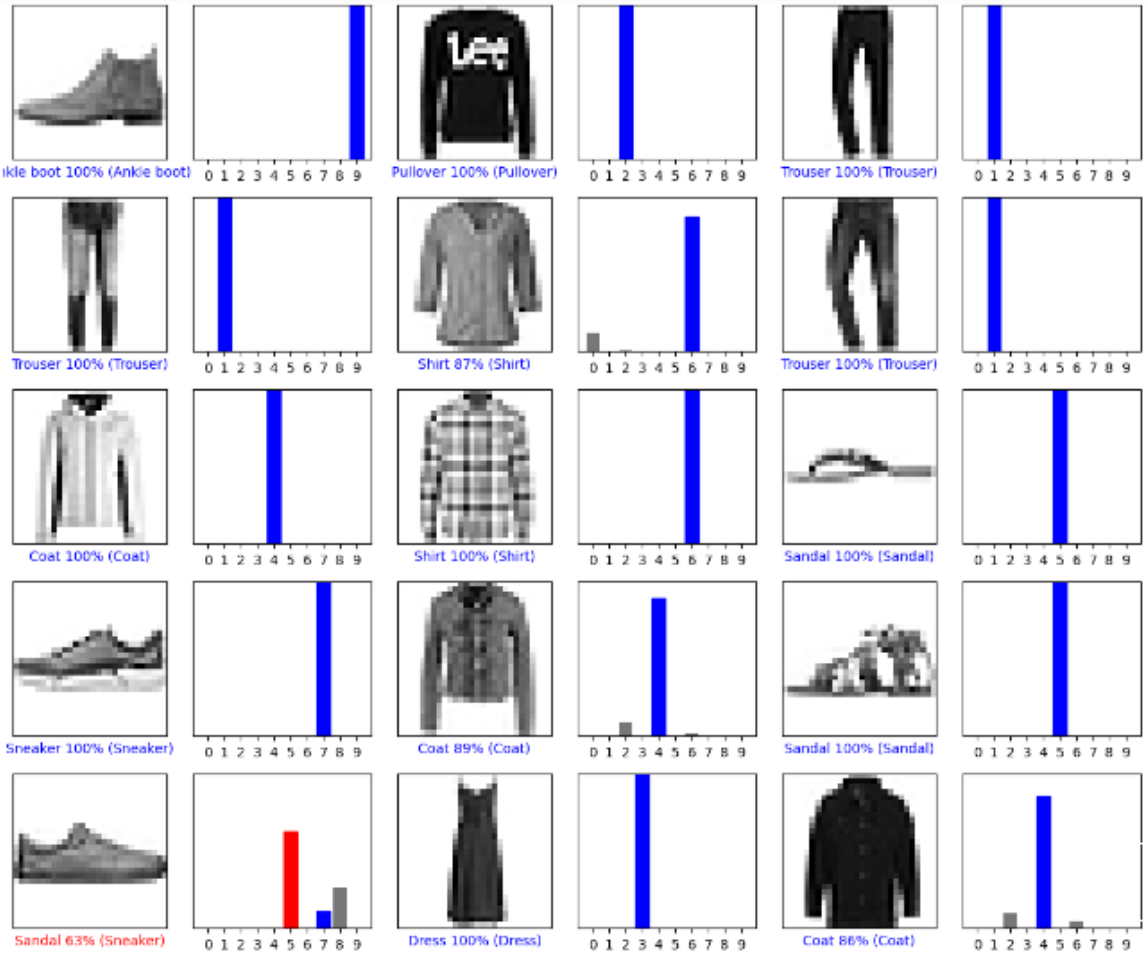
Avec le modèle formé, vous pouvez l'utiliser pour faire des prédictions sur certaines images.

Regardons la 0ème image, les prédictions et le tableau de prédiction. Les étiquettes de prédiction correctes sont bleues et les étiquettes de prédiction incorrectes sont rouges. Le nombre donne le pourcentage (sur 100) pour l'étiquette prédite.

[26]:
```
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()
```



Traçons plusieurs images avec leurs prédictions. Notez que le modèle peut se tromper même lorsqu'il est très confiant.



Utiliser le modèle entraîné

Enfin, utilisez le modèle formé pour faire une prédiction sur une seule image.

```
e [29]: img = test_images[1]

        print(img.shape)
```

```
(28, 28)
```

Les modèles tf.keras sont optimisés pour effectuer des prédictions sur un lot ou une collection d'exemples à la fois. Par conséquent, même si vous utilisez une seule image, vous devez l'ajouter à une liste :

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

      print(img.shape)
```

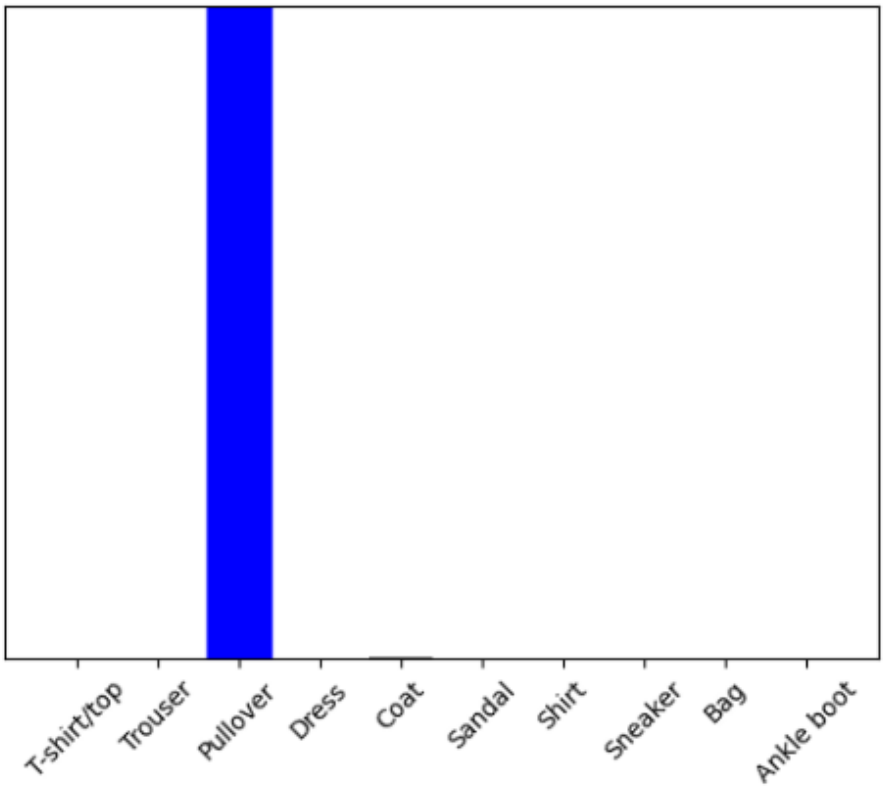
```
(1, 28, 28)
```

Prédisez maintenant le bon libellé pour cette image :

```
[31]: predictions_single = probability_model.predict(img)

print(predictions_single)
```

```
1/1 [==============================] - 4s 4s/step
[[2.8908266e-06 2.5331182e-17 9.9898690e-01 2.3197139e-10 8.7381410e-04
  7.4325745e-14 1.3639148e-04 8.7829558e-18 5.8030802e-09 8.9986256e-12]]
```

```
[32]: plot_value_array(1, predictions_single[0], test_labels)
_ = plt.xticks(range(10), class_names, rotation=45)
plt.show()
```



tf.keras.Model.predict renvoie une liste de listes, une liste pour chaque image du lot de données. Saisissez les prédictions pour notre (seule) image dans le lot :

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

```
t[33]: 2
```

Et le modèle prédit une étiquette comme prévu.