Ex No: 3 BUILD A CONVOLUTIONAL NEURAL NETWORK

AIM:

To build a simple convolutional neural network with Keras/TensorFlow.

PROCEDURE:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

PROGRAM:

```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
# Load the Fashion MNIST dataset (you can replace it with your own dataset)
fashion_mnist = tf.keras.datasets.fashion_mnist
# Split into training and test sets
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
# Reshape the images to add an extra dimension for the color channel
# CNN expects images in the format (height, width, channels)
train_images = train_images.reshape((train_images.shape[0], 28, 28, 1))
test_images = test_images.reshape((test_images.shape[0], 28, 28, 1)

# Normalize pixel values to be between 0 and 1
train_images = train_images / 255.0
test_images = test_images / 255.0
# Define the class names
```

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

# Build the CNN model
model = models.Sequential([
   # First Convolutional layer with 32 filters, kernel size of 3x3, and ReLU activation
   layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
   # First MaxPooling layer with a 2x2 pool size
   layers.MaxPooling2D((2, 2))
   # Second Convolutional layer with 64 filters
   layers.Conv2D(64, (3, 3), activation='relu'),
   # Second MaxPooling layer
   layers.MaxPooling2D((2, 2)),
   # Third Convolutional layer with 64 filters
   layers.Conv2D(64, (3, 3), activation='relu'),
   # Flatten the 3D output to 1D for the fully connected layers
   layers.Flatten()
   # Fully connected layer with 64 units
   layers.Dense(64, activation='relu'),
   # Output layer with 10 units (one for each class) with softmax activation
   layers.Dense(10)
])
# Compile the model with Adam optimizer, SparseCategoricalCrossentropy loss, and accuracy metrics
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
# Train the model on the training data
model.fit(train_images, train_labels, epochs=10)
# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print('\nTest accuracy:', test_acc)
# Create a probability model by adding a softmax layer to convert logits to probabilities
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])
```

```
# Make predictions on the test data
predictions = probability_model.predict(test_images)
# Helper function to plot predictions
def plot_image(i, predictions_array, true_label, img):
    true_label, img = true_label[i], img[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    # Remove the extra dimension for grayscale images
    img = np.squeeze(img)
    # Display the image
    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)
# Function to plot value array
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')
# Example of plotting a test image and its prediction
```

i = 0

plt.figure(figsize=(6,3))

plt.subplot(1,2,1)

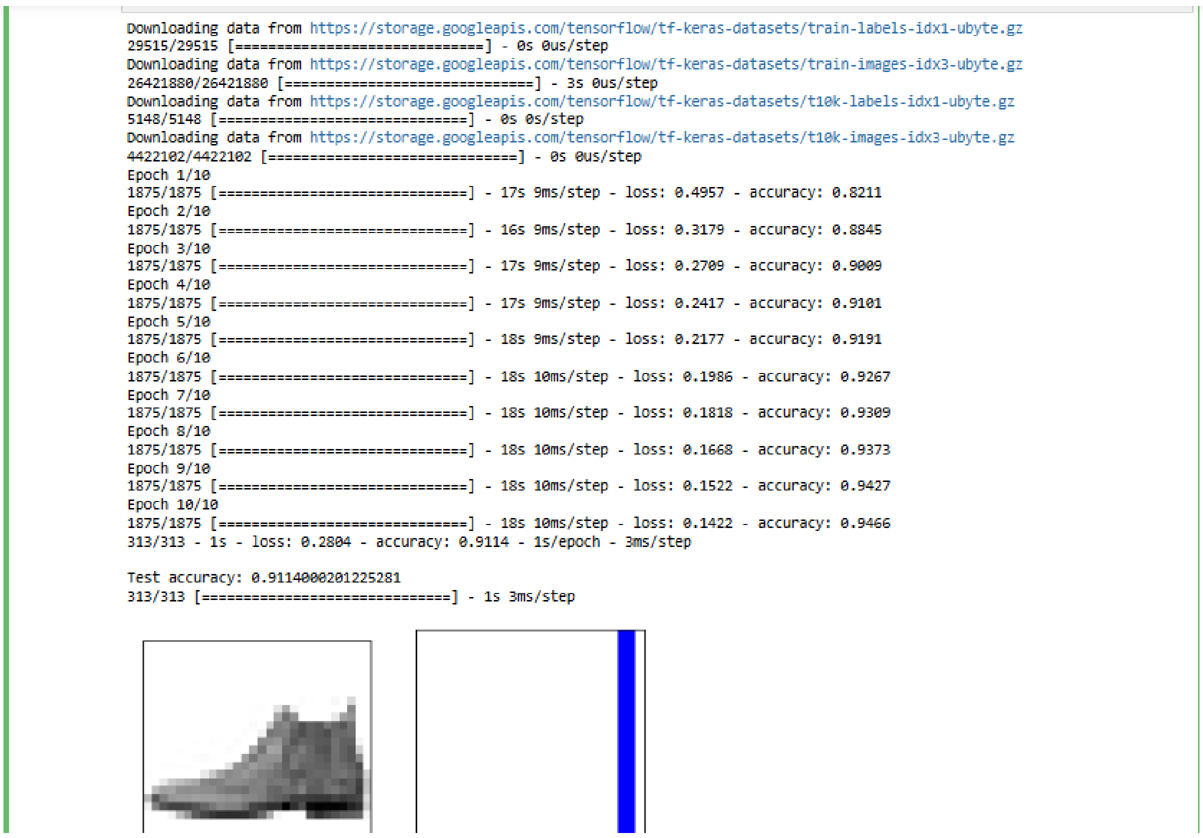
plot_image(i, predictions[i], test_labels, test_images)

plt.subplot(1,2,2)

plot_value_array(i, predictions[i], test_labels)

plt.show()

OUTPUT:



```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 3s 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 [==============================] - 0s 0us/step
Epoch 1/10
1875/1875 [==============================] - 17s 9ms/step - loss: 0.4957 - accuracy: 0.8211
Epoch 2/10
1875/1875 [==============================] - 16s 9ms/step - loss: 0.3179 - accuracy: 0.8845
Epoch 3/10
1875/1875 [==============================] - 17s 9ms/step - loss: 0.2709 - accuracy: 0.9009
Epoch 4/10
1875/1875 [==============================] - 17s 9ms/step - loss: 0.2417 - accuracy: 0.9101
Epoch 5/10
1875/1875 [==============================] - 18s 9ms/step - loss: 0.2177 - accuracy: 0.9191
Epoch 6/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.1986 - accuracy: 0.9267
Epoch 7/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.1818 - accuracy: 0.9309
Epoch 8/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.1668 - accuracy: 0.9373
Epoch 9/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.1522 - accuracy: 0.9427
Epoch 10/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.1422 - accuracy: 0.9466
313/313 - 1s - loss: 0.2804 - accuracy: 0.9114 - 1s/epoch - 3ms/step

Test accuracy: 0.9114000201225281
313/313 [==============================] - 1s 3ms/step
```

RESULT:

A simple convolutional neural network using Keras/TensorFlow is successfully build.