Ex.No: 2 Roll.No: 210701157

PYTHON PROGRAM TO BUILD A CONVOLUTIONAL NEURAL NETWORK WITH KERAS

Aim:

To build a convolutional neural network with Keras in Python.

Procedure:

1. Import TensorFlow and Keras modules for building and training the model.
2. Load the CIFAR-10 dataset consisting of 60,000 32x32 color images across 10 classes.
3. Normalize the pixel values of the training and testing images to be between 0 and 1.
4. Print the shape of the training and test images and labels to verify the dataset dimensions.
5. Create a Sequential CNN model starting with a 32-filter Conv2D layer followed by MaxPooling.
6. Add two more Conv2D layers with 64 filters each, followed by MaxPooling and activation.
7. Flatten the feature map and add a Dense layer with 64 units and an output layer with 10 units.
8. Compile the model using Adam optimizer, Sparse Categorical Crossentropy loss, and accuracy metrics.
9. Train the model for 10 epochs with validation on test data and store the training history.
10. Evaluate the model's accuracy on the test set and visualize the accuracy and loss during training using plots.

Code:

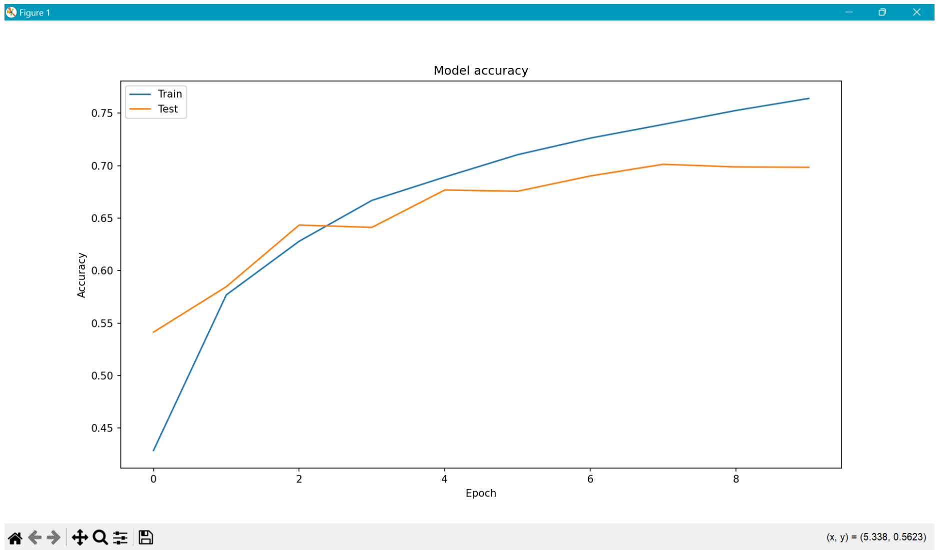
```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt

# Load the CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) =
tf.keras.datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

# Print the shape of the dataset
print(f"Training images shape: {train_images.shape}")
print(f"Training labels shape: {train_labels.shape}")
print(f"Test images shape: {test_images.shape}")
print(f"Test labels shape: {test_labels.shape}")

model = models.Sequential([
    # Convolutional layer 1
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    # Convolutional layer 2
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    # Convolutional layer 3
    layers.Conv2D(64, (3, 3), activation='relu'),

    # Flatten and Fully Connected (Dense) Layers
```

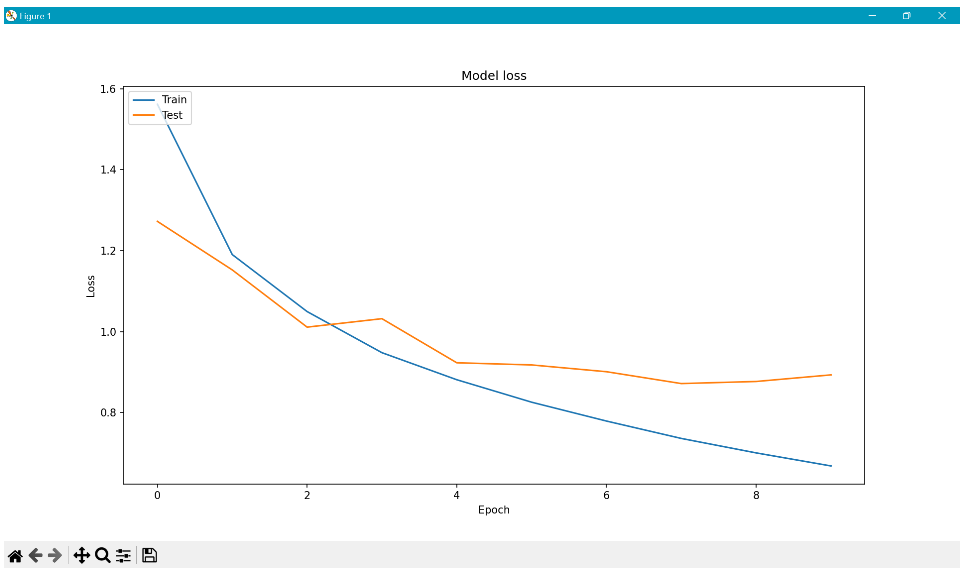
```
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10)  # 10 classes
])

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

history = model.fit(
    train_images,
    train_labels,
    epochs=10,
    validation_data=(test_images, test_labels)
)

test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f'\nTest accuracy: {test_acc}')

predictions = model.predict(test_images)
print(f'Predictions shape: {predictions.shape}')

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper left')
```

```
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

Output:

```
Command Prompt - python (

t_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the fi rst layer in the model in
stead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2024-08-16 12:54:21.597114: I tensorfl ow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instru
ctions in performance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler fl ags.
Epoch 1/10
1563/1563 -------------------- 11s 6ms/step - accuracy: 0.3290 - loss: 1.8035 - val_accuracy: 0.5414 - val_loss: 1.2722
Epoch 2/10
1563/1563 -------------------- 11s 7ms/step - accuracy: 0.5651 - loss: 1.2240 - val_accuracy: 0.5847 - val_loss: 1.1523
Epoch 3/10
1563/1563 -------------------- 11s 7ms/step - accuracy: 0.6180 - loss: 1.0743 - val_accuracy: 0.6434 - val_loss: 1.0110
Epoch 4/10
1563/1563 -------------------- 11s 7ms/step - accuracy: 0.6614 - loss: 0.9544 - val_accuracy: 0.6412 - val_loss: 1.0319
Epoch 5/10
1563/1563 -------------------- 10s 6ms/step - accuracy: 0.6855 - loss: 0.8848 - val_accuracy: 0.6768 - val_loss: 0.9229
Epoch 6/10
1563/1563 -------------------- 10s 7ms/step - accuracy: 0.7126 - loss: 0.8208 - val_accuracy: 0.6756 - val_loss: 0.9176
Epoch 7/10
1563/1563 -------------------- 11s 7ms/step - accuracy: 0.7287 - loss: 0.7728 - val_accuracy: 0.6902 - val_loss: 0.9008
Epoch 8/10
1563/1563 -------------------- 10s 6ms/step - accuracy: 0.7412 - loss: 0.7281 - val_accuracy: 0.7013 - val_loss: 0.8715
Epoch 9/10
1563/1563 -------------------- 10s 7ms/step - accuracy: 0.7561 - loss: 0.6937 - val_accuracy: 0.6988 - val_loss: 0.8767
Epoch 10/10
1563/1563 -------------------- 11s 7ms/step - accuracy: 0.7716 - loss: 0.6519 - val_accuracy: 0.6984 - val_loss: 0.8930
313/313 - 1s - 4ms/step - accuracy: 0.6984 - loss: 0.8930

Test accuracy: 0.6984000205993652
313/313 -------------------- 1s 2ms/step
Predictions shape: (10000, 10)
```





Result:

Thus, to build a convolutional neural network using keras has been completed successfully.