NANDHA ENGINEERING COLLEGE, AUTONOMOUS, ERODE -52

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ASSIGNMENT -2

ACADEMIC YEAR: 2024-2025

Register Number	:	22cs095, 22cs096
Name	:	Sowmiya Devi P, Sowmya A
COURSE CODE & NAME	:	**22CSX01 & DEEP LEARNING**
CLASS /SEM	:	**III- B.E(CSE) / V**

TEAM – 19

TOPIC	MARKS
You are tasked with improving the computational efficiency of a CNN model for a mobile application. How would you incorporate depth wise separable convolutions?	

Student signature **Faculty Signature**

1. You are tasked with improving the computational efficiency of a CNN model for a mobile application. How would you incorporate depth wise separable convolutions?

Aim:

To improve the computational efficiency of a Convolutional Neural Network (CNN) model for a mobile application by incorporating depth wise separable convolutions. This approach reduces the computational cost and model size while maintaining or improving performance, making it suitable for resource-constrained environments like mobile devices.

Objectives:

1. To understand the limitations of traditional CNN architectures for mobile applications in terms of computational cost and energy efficiency.
2. To implement depth wise separable convolutions as a replacement for standard convolution layers in the CNN architecture.
3. To analyze the impact of depth wise separable convolutions on model performance, inference time, and memory usage.
4. To validate the optimized CNN model on mobile devices or simulators for practical usability.

Significance:

Depthwise separable convolutions decompose the standard convolution into two steps:

1. Depthwise Convolution: Applies a single filter to each input channel independently.
2. Pointwise Convolution: Applies a 1x1 convolution to combine the outputs of the depthwise convolution, creating feature maps.

This decomposition significantly reduces the number of parameters and computations required, making it ideal for mobile and edge applications where computational resources are limited.

Coding:

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, DepthwiseConv2D, Dense, Flatten, MaxPooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(-1, 28, 28, 1).astype("float32") / 255.0
x_test = x_test.reshape(-1, 28, 28, 1).astype("float32") / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
# Define a CNN with depthwise separable convolutions
```

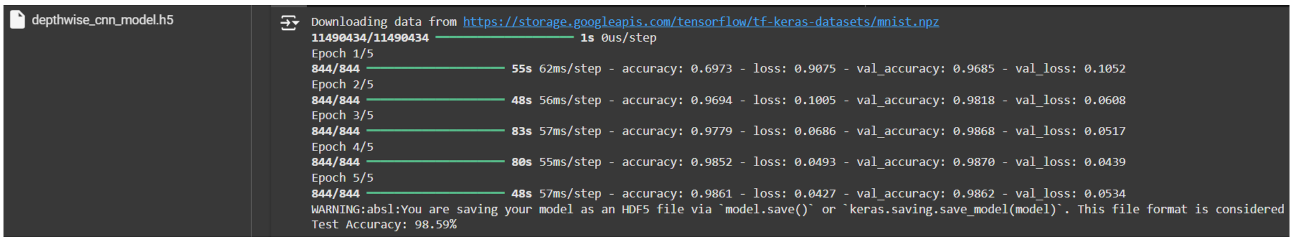
```
def create_model():
    input_layer = Input(shape=(28, 28, 1))
    # First layer with standard convolution
    x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_layer)
    x = MaxPooling2D((2, 2))(x)
    # Depthwise separable convolution
    x = DepthwiseConv2D((3, 3), activation='relu', padding='same')(x)
    x = Conv2D(64, (1, 1), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2))(x)
    # Another Depthwise separable convolution
    x = DepthwiseConv2D((3, 3), activation='relu', padding='same')(x)
    x = Conv2D(128, (1, 1), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2))(x)

    # Flatten and Dense layers
    x = Flatten()(x)
    x = Dense(128, activation='relu')(x)
    output_layer = Dense(10, activation='softmax')(x)

    model = Model(inputs=input_layer, outputs=output_layer)
    return model
# Create and compile the model
model = create_model()
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=5, batch_size=64, validation_split=0.1)

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print(f"Test Accuracy: {test_acc * 100:.2f}%")
model.save("depthwise_cnn_model.h5")
```

Output:

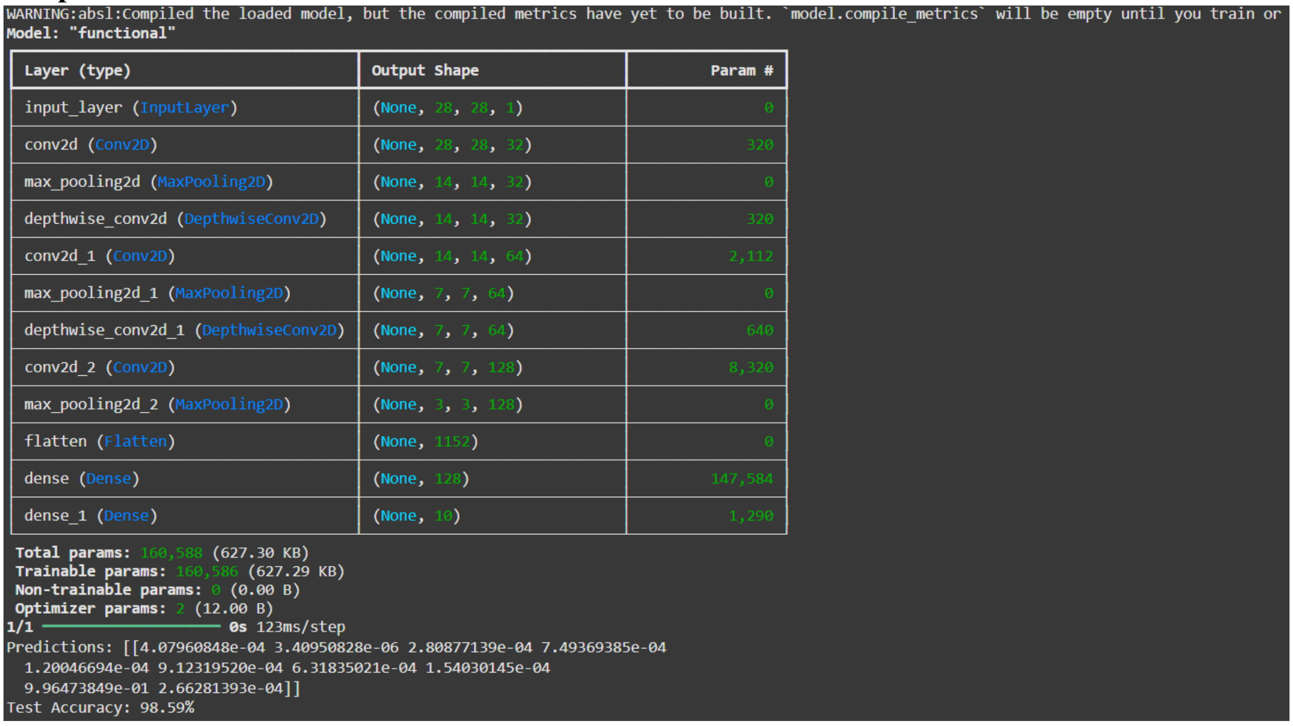


```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
Epoch 1/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 55s 62ms/step - accuracy: 0.6973 - loss: 0.9075 - val_accuracy: 0.9685 - val_loss: 0.1052
Epoch 2/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 48s 56ms/step - accuracy: 0.9694 - loss: 0.1005 - val_accuracy: 0.9818 - val_loss: 0.0608
Epoch 3/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 83s 57ms/step - accuracy: 0.9779 - loss: 0.0686 - val_accuracy: 0.9868 - val_loss: 0.0517
Epoch 4/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 80s 55ms/step - accuracy: 0.9852 - loss: 0.0493 - val_accuracy: 0.9870 - val_loss: 0.0439
Epoch 5/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 48s 57ms/step - accuracy: 0.9861 - loss: 0.0427 - val_accuracy: 0.9862 - val_loss: 0.0534
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
Test Accuracy: 98.59%
```

```
import tensorflow as tf
# Load the saved model
model = tf.keras.models.load_model("depthwise_cnn_model.h5")
# Verify the model structure
model.summary()
# Example: Make predictions on new data
import numpy as np
# Create a dummy input for testing (e.g., a single MNIST image)
dummy_input = np.random.rand(1, 28, 28, 1)  # Shape: (batch_size, height, width, channels)
```

```
# Predict
predictions = model.predict(dummy_input)
print("Predictions:", predictions
# Example: Evaluate the model on test data (if available)
# Assuming you still have x_test and y_test loaded
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print(f"Test Accuracy: {test_acc * 100:.2f}%")
```

Output:



```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or
Model: "functional"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ input_layer (InputLayer)             │ (None, 28, 28, 1)           │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d (Conv2D)                      │ (None, 28, 28, 32)          │             320 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d (MaxPooling2D)         │ (None, 14, 14, 32)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ depthwise_conv2d (DepthwiseConv2D)   │ (None, 14, 14, 32)          │             320 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_1 (Conv2D)                    │ (None, 14, 14, 64)          │           2,112 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_1 (MaxPooling2D)       │ (None, 7, 7, 64)            │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ depthwise_conv2d_1 (DepthwiseConv2D) │ (None, 7, 7, 64)            │             640 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_2 (Conv2D)                    │ (None, 7, 7, 128)           │           8,320 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_2 (MaxPooling2D)       │ (None, 3, 3, 128)           │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten (Flatten)                    │ (None, 1152)                │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense (Dense)                        │ (None, 128)                 │         147,584 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_1 (Dense)                      │ (None, 10)                  │           1,290 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 160,588 (627.30 KB)
 Trainable params: 160,586 (627.29 KB)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 2 (12.00 B)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 123ms/step
Predictions: [[4.07960848e-04 3.40950828e-06 2.80877139e-04 7.49369385e-04
  1.20046694e-04 9.12319520e-04 6.31835021e-04 1.54030145e-04
  9.96473849e-01 2.66281393e-04]]
Test Accuracy: 98.59%
```

RESULT:

The CNN model successfully improving the computational efficiency of a CNN model for a mobile application