Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_bot] THANOS

Experiment 8: Build and train a ConvNet in Pytorch for a classification problem.

Aim/Objective: To build and train a Convolutional Neural Network (ConvNet) in TensorFlow for a classification problem. The experiment aims to provide hands-on experience in implementing and training a ConvNet without using high-level libraries, emphasizing the manual construction of the neural network architecture and the training process.

Description: In this lab experiment, the goal is to implement a ConvNet using Pytorch for a classification task. The experiment involves constructing the ConvNet architecture, defining operations for forward and backward passes, setting up data preprocessing, and training the model.

Pre-Requisites: Basic knowledge of Neural Networks, TensorFlow Basics, Python Programming, Gradient Descent and Backpropagation, Convolutional Neural Networks.

Pre-Lab:

1. What is the primary purpose of a convolutional layer in a ConvNet, and how does it contribute to feature extraction?

Extracts features by applying filters to detect patterns like edges and textures.

2. Briefly explain the role of pooling layers in ConvNets. How do they contribute to spatial downsampling?

Reduces spatial dimensions, improves efficiency, and retains important features (e.g., max pooling).

3. What is the role of fully connected layers in ConvNets, and how do they contribute to the final classification decision?

Combines extracted features for final classification using activation functions like Softmax.

4. How is backpropagation applied in ConvNets during the training process, considering both convolutional and fully connected layers?

Updates weights by computing gradients in both convolutional and fully connected layers using techniques like gradient descent.

In-Lab:

Program 1: Write a Python script to manually construct the architecture of a ConvNet, including convolutional layers, pooling layers, fully connected layers, and activation functions.

Procedure/Program:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation

def build_cnn():
    model = Sequential()

    model.add(Conv2D(filters=32, kernel_size=(3,3), padding='same', input_shape=(64, 64, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Conv2D(filters=64, kernel_size=(3,3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Flatten())

    model.add(Dense(128))
    model.add(Activation('relu'))

    model.add(Dense(10))
    model.add(Activation('softmax'))

    return model

cnn_model = build_cnn()

cnn_model.summary()
```

- **Data and Results:**

Data

The dataset consists of images with dimensions 64x64x3 for classification.

Result

The model successfully classifies images into 10 different categories.

- **Analysis and Inferences:**

Analysis

The CNN architecture extracts features through convolutional and pooling layers.

Inferences

Increasing layers improves accuracy but may lead to overfitting issues.

Sample VIVA-VOCE Questions (In-Lab):

1. How does data augmentation contribute to improving the robustness of a ConvNet for image classification tasks?

Data Augmentation enhances model robustness by generating variations (rotation, flipping, scaling) to reduce overfitting and improve generalization.

2. How does the choice of stride size in convolutional layers impact the spatial dimensions of the feature maps and, consequently, the model's receptive field?

Stride Size controls feature map size—larger strides reduce dimensions and details, while smaller strides retain more information but increase computation. It also affects the receptive field.

3. What is the role of batch normalization in ConvNets, and how does it contribute to training stability and faster convergence?

Batch Normalization stabilizes training by normalizing activations, speeding up convergence, and reducing sensitivity to hyperparameters.

4. Can you discuss other applications where ConvNets have demonstrated success, such as object detection or segmentation?

ConvNet Applications include **object detection** (YOLO, Faster R-CNN), **segmentation** (U-Net, Mask R-CNN), facial recognition, and medical imaging.

Post-Lab:

Program 2: Implement the training script for the ConvNet using TensorFlow, including forward and backward passes, gradient descent optimization, and evaluation on a classification dataset.

Procedure/Program:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()

x_train, x_test = x_train / 255.0, x_test / 255.0

y_train, y_test = y_train.flatten(), y_test.flatten()

def create_model():
  model = keras.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),

    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
  ])
```

```
    return model

model = create_model()

model.compile(
    optimizer=keras.optimizers.Adam(),
    loss=keras.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy']
)

history = model.fit(
    x_train, y_train, epochs=10, batch_size=64, validation_data=(x_test, y_test)
)

test_loss, test_acc = model.evaluate(x_test, y_test)

print(f"Test Accuracy: {test_acc:.4f}")

plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

- **Data and Results:**

Data

The CIFAR-10 dataset contains 60,000 images in 10 categories.

Result

The trained ConvNet achieved a certain accuracy on test data.

- **Analysis and Inferences:**

Analysis

The model performance improved over epochs, showing learning progression.

Inferences

Higher convolutional layers contributed to better feature extraction accuracy.

Evaluator Remark (if Any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.