**Part A: Theoretical Concepts**

### ****1. Activation Functions****

Define and compare the following activation functions: Sigmoid, ReLU, Tanh, and Leaky ReLU.

#### (a) **Sigmoid**:

* **Formula**: σ(x)=11+e−x\sigma(x) = \frac{1}{1 + e^{-x}}σ(x)=1+e−x1​
* **Range**: (0,1)(0, 1)(0,1)
* **Use Case**: Used in binary classification tasks.
* **Limitation**: Vanishing gradients during backpropagation for large or small values of xxx.

#### (b) **ReLU (Rectified Linear Unit)**:

* **Formula**: f(x)=max⁡(0,x)f(x) = \max(0, x)f(x)=max(0,x)
* **Range**: [0,∞)[0, \infty)[0,∞)
* **Use Case**: Default activation for deep networks.
* **Limitation**: Dead neurons (outputs zero for negative inputs).

#### (c) **Tanh**:

* **Formula**: tanh⁡(x)=ex−e−xex+e−x\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}tanh(x)=ex+e−xex−e−x​
* **Range**: (−1,1)(-1, 1)(−1,1)
* **Use Case**: Normalized data-centered activations.
* **Limitation**: Vanishing gradients.

#### (d) **Leaky ReLU**:

* **Formula**: f(x)=max⁡(0.01x,x)f(x) = \max(0.01x, x)f(x)=max(0.01x,x)
* **Range**: (−∞,∞)(-\infty, \infty)(−∞,∞)
* **Use Case**: Solves the dead neurons problem.
* **Limitation**: Requires manual tuning of the slope for negative inputs.

**2. Discussion of Optimization Algorithms**

* **Comparison**:
  + **SGD (Stochastic Gradient Descent)**:
    - Simple, faster for large datasets.
    - Sensitive to learning rate; may converge slowly.
  + **Adam (Adaptive Moment Estimation)**:
    - Combines momentum and adaptive learning rates.
    - Suitable for sparse gradients.
  + **RMSprop (Root Mean Square Propagation)**:
    - Adjusts learning rate using recent gradient magnitudes.
    - Performs well in RNNs and non-stationary problems.
* **Learning Rate Impact**:
  + High learning rate: Fast but may overshoot the minimum.
  + Low learning rate: Stable but slow convergence.
  + Modern optimizers adapt the learning rate dynamically.

**Part B: Practical Implementation**

**1. Data Preprocessing :Download and preprocess the CIFAR-10 dataset**

import tensorflow as tf

from tensorflow.keras.datasets import cifar10

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

data\_augmentation = tf.keras.Sequential([

tf.keras.layers.RandomFlip("horizontal"),

tf.keras.layers.RandomRotation(0.1)])

**2. Model Design**

* **Design a CNN**:
  + At least 3 convolutional layers and 2 fully connected layers.
  + Include regularization

**Code:**

from tensorflow.keras import models, layers

model = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)),

layers.MaxPooling2D((2, 2)),

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

layers.MaxPooling2D((2, 2)),

layers.Conv2D(128, (3, 3), activation='relu'),

layers.Flatten(),

layers.Dense(128, activation='relu'),

layers.Dropout(0.5),

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

**3. Model Training**

* **Compile and train the model**:Use early stopping or learning rate scheduling if necessary.

**Code:**

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

early\_stopping = tf.keras.callbacks.EarlyStopping(monitor='val\_loss', patience=5)

history = model.fit( x\_train, y\_train, epochs=30,

validation\_data=(x\_test, y\_test),

callbacks=[early\_stopping])

**4. Model Evaluation**

* Evaluate the model on the test set.
* Generate a confusion matrix.

**Code:**

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

print(f"Test Accuracy: {test\_acc}")

y\_pred = model.predict(x\_test).argmax(axis=1)

y\_true = y\_test.argmax(axis=1)

cm = confusion\_matrix(y\_true, y\_pred)

ConfusionMatrixDisplay(cm).plot()

plt.show()

**5. Error Analysis and conclusion**

* Identify errors using the confusion matrix.
* **Example of errors**:
  1. Class A misclassified as Class B .
  2. Poor performance on smaller objects.
  3. Misclassification in overlapping classes.

**Proposed Solutions for error:**

* Increase training data diversity using augmentation.
* Use a pre-trained model for transfer learning.
* Fine-tune hyperparameters or increase model complexity.