Part A: Theoretical Concepts

1. Activation Functions

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

(a) Sigmoid:

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

(b) ReLU (Rectified Linear Unit):

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

(c) Tanh:

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

(d) Leaky ReLU:

- **Formula**: $f(x) = \max(0.01x, x)$
- **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.