Statistical Deep Learning (MT7042) - Project 2

Instructions: This project consists of only one problem worth a total of 100 points. Submit your completed solution as a Jupyter Notebook (`.ipynb`) if you are using Python, or an R Markdown file (`.Rmd`) if you are using R.

Additional Guidelines:

- You are encouraged to discuss the assignment. However, all solutions are to be written individually.
- Ensure that all code is runnable.

Problem 1

For Python users, please follow this tutorial, and for R users, please follow this tutorial. Python users can download the Jupyter notebook directly, while R users can create a new Markdown file and copy the code blocks provided in the tutorial.

- **Task 1**: Follow the tutorial step-by-step.
- **Task 2 (20 p)**: Repeat Task 1 using the `tanh` activation. Compare the performance of the `tanh` activation function with `ReLU` in the CNN. Additionally, discuss why `tanh` might yield better or worse results than `ReLU`. **Note:** You may need to increase the number of epochs if the loss does not seem to converge.
- **Task 3 (20 p)**: Refer to the table summarizing the output of the convolutional layers and explain why the number of channels is set to 32 and 64, considering that the original image has only 3 color channels (i.e., R, G, and B).
- **Task 4 (20 p)**: Extend the code in Task 1 (with `ReLU` activation) by incorporating batch normalization (BN) in your CNN.
  - Investigate the effect of BN on the model's loss and test error, as done in Task 1.
  - Discuss the possible reasons for any increase or decrease in performance when using batch normalization.

  Note: Refer to Sections 8.5.1 and 8.5.2 of the book *Dive into Deep Learning* (https://d2l.ai/) to understand the concepts before modifying your code.
- **Task 5 (20 p)**: Apply data augmentation techniques (e.g., random flips, rotations, and cropping) to your training set based on the model configuration in Task 1.
  - Visualize a few augmented examples (e.g., ~ 5 images) to show how the transformations alter the original training images.

  - Explain why data augmentation can help reduce overfitting and improve the generalization power of the model.
  - Compare the test accuracy and loss with and without augmentation.

- **Task 6 (20 p)**: Assess the role of color information in feature extraction by converting the input images to grayscale. Retrain the CNN with the model configuration from Task 1 and compare its performance to training on RGB images.

  - Python:

```python
# !pip install scikit-image
from skimage.color import rgb2gray
import numpy as np

# Convert train and test images to grayscale
train_images_gray = rgb2gray(train_images)
test_images_gray = rgb2gray(test_images)

# Add channel dimension for compatibility with CNNs
train_images_gray = train_images_gray[..., np.newaxis]
test_images_gray = test_images_gray[..., np.newaxis]
```

Listing 1: Convert CIFAR-10 images to grayscale in Python

  - R

```r
library(magick)
rgb_to_grayscale <- function(images) {
  lapply(1:dim(images)[1], function(i) {
    # Convert array to magick image
    img <- image_read(images[i,,,])
    img_gray <- image_convert(img, colorspace = "gray")
    as.numeric(image_data(img_gray))
  })
}
# Apply grayscale conversion to training and test images
ctrain_x_gray <- rgb_to_grayscale(cifar10$train$x)
ctest_x_gray <- rgb_to_grayscale(cifar10$test$x)
# Convert the list back to an array
ctrain_x_gray <- array(unlist(ctrain_x_gray),
    dim = c(50000, 32, 32, 1))
ctest_x_gray <- array(unlist(ctest_x_gray),
    dim = c(10000, 32, 32, 1))
```

Listing 2: Convert CIFAR-10 images to grayscale in R