COSC 2753 | Machine Learning

Week 10 Lab Exercises: **Neural Networks (MLP)**

Introduction

In this lab you will be:

1. Learning how to use Keras API (in tensorflow library).
2. Implement simple NN to classify images in the CIFAR-10 dataset.

The CIFAR-10 dataset

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:

drawing

The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

In this lab we will only use a subset of CIFAR10 data that has 10000 images.

Reading Data

The data is available on Canvas. You can upload it to the notebook work environment and unzip using the following code.

```
import zipfile
with zipfile.ZipFile('CIFAR10_Lab9.zip', 'r') as zip_ref:
    zip_ref.extractall('./')

```

The dataset consists of the images and a csv file. The labels and the image paths are in the CSV file.

Lets randomly split the data into train/val/test

```
import numpy as np
from sklearn.model_selection import train_test_split
import pandas as pd

data = pd.read_csv('./CIFAR_Data.csv')

train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)
train_data, val_data = train_test_split(train_data, test_size=0.25, random_state=42)

print("Train data : {}, Val Data: {}, Test Data: {}".format(train_data.shape[0], val_data.shape[0], test_data.shape[0]))
```

```
Train data : 6000, Val Data: 2000, Test Data: 2000
```

EDA

> ☞ **Task: Do the EDA and identify the suitable performance measure.**

> ☞ **Task: Plot some images from the dataset and observe.**

```
print(train_data.columns)
```

```
Index(['ImgPath', 'Class'], dtype='object')
```

+ Code + Text

Exploratory Data Analysis (EDA) is a crucial step in understanding your CIFAR-10 dataset. For your image dataset, EDA can involve:

1. **Viewing Images**: Display a sample of images from each class to understand their visual characteristics. This will give you a sense of the variability within and across classes.
2. **Class Distribution**: Check if the dataset is balanced across different classes. This can be done by plotting the distribution of classes in your training, validation, and test sets.

1. **Viewing Images**: Display a sample of images from each class to understand their visual characteristics. This will give you a sense of the variability within and across classes.

```
import matplotlib.pyplot as plt
import numpy as np

# Update to use the correct column name for classes
classes = np.unique(train_data['Class'])

# Number of samples to display from each class
num_samples = 5

# Plotting
fig, axes = plt.subplots(len(classes), num_samples, figsize=(num_samples*2, len(classes)*2))

for i, cls in enumerate(classes):
    # Selecting the first 'num_samples' images of class 'cls'
    class_images = train_data[train_data['Class'] == cls].iloc[:num_samples]
    for j in range(num_samples):
        # Load image from 'ImgPath' and reshape
        # You might need to adjust this part based on how you load images
        img = plt.imread(class_images.iloc[j]['ImgPath'])
        axes[i, j].imshow(img)
        axes[i, j].axis('off')
        if j == 0:
            axes[i, j].set_title(f"Class: {cls}")

plt.tight_layout()
plt.show()

```

The provided Python code is used to display a grid of images from a dataset, with each row in the grid corresponding to a different class of images. The code uses the `matplotlib` library for creating the grid and displaying the images, and the `numpy` library for handling the classes.

The variable `classes` is assigned the unique values in the 'Class' column of the `train_data` DataFrame, which represents the different classes of images in the dataset.

The variable `num_samples` is set to 5, indicating that five images from each class will be displayed.

The `plt.subplots` function is then used to create a grid of subplots with a number of rows equal to the number of classes and a number of columns equal to `num_samples`. The size of the figure is set to be proportional to the number of rows and columns.

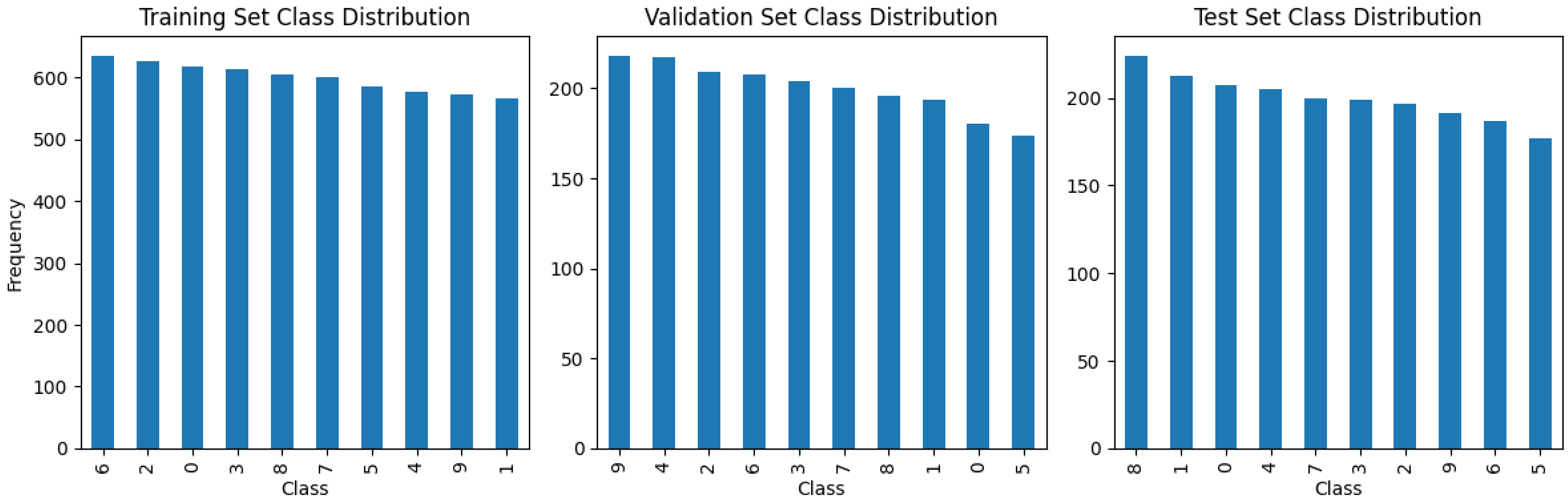
The code then enters a loop over the classes. For each class, it selects the first `num_samples` rows from `train_data` where the 'Class' column is equal to the current class. This subset of `train_data` is stored in the `class_images` DataFrame.

Inside this loop, there is another loop over the range `num_samples`. For each iteration, it reads the image file specified in the 'ImgPath' column of the `class_images` DataFrame, displays the image in the corresponding subplot using the `imshow` function, and turns off the axis. If the current sample is the first one (i.e., `j` is 0), it also sets the title of the subplot to the name of the class.

After both loops have finished, the `plt.tight_layout` function is called to adjust the spacing between the subplots, and `plt.show` is used to display the figure. The result is a grid of images where each row corresponds to a different class, and each column corresponds to a different sample from that class.

2. **Class Distribution**: Check if the dataset is balanced across different classes. This can be done by plotting the distribution of classes in your training, validation, and test sets.

```
import matplotlib.pyplot as plt

# Count the occurrence of each class in each dataset
train_class_counts = train_data['Class'].value_counts()
val_class_counts = val_data['Class'].value_counts()
test_class_counts = test_data['Class'].value_counts()

# Create a bar plot for each dataset
plt.figure(figsize=(12, 4))

plt.subplot(1, 3, 1)
train_class_counts.plot(kind='bar')
plt.title('Training Set Class Distribution')
plt.xlabel('Class')
plt.ylabel('Frequency')

plt.subplot(1, 3, 2)
val_class_counts.plot(kind='bar')
plt.title('Validation Set Class Distribution')
plt.xlabel('Class')

plt.subplot(1, 3, 3)
test_class_counts.plot(kind='bar')
plt.title('Test Set Class Distribution')
plt.xlabel('Class')

plt.tight_layout()
plt.show()

```



Performance metrics like accuracy_score, precision, recall, F1 score, or ROC-AUC could be considered depending on the specific requirements and characteristics of the dataset, such as class imbalance or the importance of false positives versus false negatives.

› Model development

As discussed in Week 4-5, the typical ML model development process consists of 4 steps, lets go through each and see how it is done.

1. **Determine your goals**: Performance metric and target value. Problem dependent.

2. **Setup the experiment**: Setup the test/validation data, visualisers and debuggers needed to determine bottlenecks in performance (overfitting/under-fitting, feature importance).
3. **Default Baseline Model**: Identify the components of end-to-end pipeline including - Baseline Models, cost functions, optimisation.
4. **Make incremental changes**: Repeatedly make incremental changes such as gathering new data, adjusting hyper-parameters, or changing algorithms, based on specific findings from your instrumentation.

So far we conducted the EDA and accordingly we can decide that a performance measure such as Accuracy is adequate for this task. This is justified by the observation that there is no label imbalance in our dataset and the task is multi-class classification. According to the CIFAR website others have achieved around 82% accuracy for the full dataset. Therefore we can set that as the target performance value. However this might be unrealistic given:

- We are only using a subset of data in this lab (10000/50000) to train the model.
- We are using a traditional neural network (not CNN).
- No data augmentation.

For a traditional NN a 50% accuracy target will be more realistic.

We have also identified the train/test/val splits and ready to do the experiments. In developing a neural network, we can use the learning curves to identify the next action to take. so let's write a simple function to plot the learning curve of a NN training process. This will be our diagnostic tool.

[ ] ↪ *28 cells hidden*

Testing the final model

Now lets test our final model on the test data.

First we will create a data generator to get the test data

[ ] ↪ *6 cells hidden*