BU.330.760 Deep Learning with Unstructured Data
Lab 5. TensorFlow and Convolutional Neural Network

Learning Goal: practice using TensorFlow tool to train a convolutional neural network for image classification
Required Skills: knowledge on convolutional neural network and TensorFlow

1. We will use TensorFlow (https://www.tensorflow.org) in lab 5, which is an open source library for machine learning. We will use a filtered version of Dogs vs Cats dataset (https://www.kaggle.com/c/dogs-vs-cats/data) from Kaggle for this lab. The original dataset contains 25,000 images of dogs and cats.
2. **If you are using Anaconda**, we've already installed TensorFlow in lab 4 since Keras on Anaconda uses TensorFlow as the underlying package, and installing Keras results in installing TensorFlow as well. Therefore, you simply need to go to the environment you built in lab 4 and lunch Jupyter notebook.

   If you are using Google CoLab, TensorFlow package is already available on it.

3. In Jupyter notebook or Google CoLab, build the following scripts
   a. Import required packages. The **os** package is used to read files and directory structure.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
```

   b. First, let's download the dataset. After extracting the zipped content, get the proper file paths for the training and validation set. We will have directories for training cat pictures, training dog pictures, validation cat pictures, and validation dog pictures.

```
_URL = 'https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip'
path_to_zip = tf.keras.utils.get_file('cats_and_dogs.zip', origin=_URL, extract=True)
PATH = os.path.join(os.path.dirname(path_to_zip), 'cats_and_dogs_filtered')

train_dir = os.path.join(PATH, 'train')
validation_dir = os.path.join(PATH, 'validation')
train_cats_dir = os.path.join(train_dir, 'cats')
train_dogs_dir = os.path.join(train_dir, 'dogs')
validation_cats_dir = os.path.join(validation_dir, 'cats')
validation_dogs_dir = os.path.join(validation_dir, 'dogs')
```

c. There are 1000 training cat pictures, 1000 training dog pictures, 500 validation cat pictures, and 500 validation dog pictures in total.

```
num_cats_tr = len(os.listdir(train_cats_dir))
num_dogs_tr = len(os.listdir(train_dogs_dir))
num_cats_val = len(os.listdir(validation_cats_dir))
num_dogs_val = len(os.listdir(validation_dogs_dir))
total_train = num_cats_tr + num_dogs_tr
total_val = num_cats_val + num_dogs_val

print('total training cat images:', num_cats_tr)
print('total training dog images:', num_dogs_tr)
print('total validation cat images:', num_cats_val)
print('total validation dog images:', num_dogs_val)
print("--")
print("Total training images:", total_train)
print("Total validation images:", total_val)
```

d. Then let's set up variables, such as batch size and number of training epochs.

```
batch_size = 128
epochs = 15
IMG_HEIGHT = 150
IMG_WIDTH = 150
```

e. Use **ImageDataGenerator** class to read images and preprocess them into proper tensors. We will again rescale the tensors from 0-255 to 0-1, as neural networks prefer to deal with small input values. Then use its **flow_from_directory** method to load images from the disk, applies rescaling, and resizes the images into the required dimensions.

```
train_image_generator = ImageDataGenerator(rescale=1./255)
validation_image_generator = ImageDataGenerator(rescale=1./255)

train_data_gen = train_image_generator.flow_from_directory(batch_size=batch_size,
directory=train_dir, shuffle=True, target_size=(IMG_HEIGHT, IMG_WIDTH),
class_mode='binary')

val_data_gen = validation_image_generator.flow_from_directory(batch_size=batch_size,
directory=validation_dir,target_size=(IMG_HEIGHT, IMG_WIDTH),class_mode='binary')
```

f. Visualize the training images using **next** function, which returns a batch of images from the training dataset. Then a self-defined function will be used to plot five of the images.

```
sample_training_images, _ = next(train_data_gen)
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip( images_arr, axes):
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()
plotImages(sample_training_images[:5])
```

g. Now we can build the CNN model, which consists of three convolution blocks with a max pool layer in each of them. There's a fully connected layer with 512 units on top of it that is activated by a **relu** activation function.

   We will use **adam** optimizer and binary cross entropy loss function, together with **accuracy** as the metrics to view training and validation accuracy for each training epoch.

```
model = Sequential([
    Conv2D(16, 3, padding='same', activation='relu', input_shape=(IMG_HEIGHT, 
IMG_WIDTH ,3)),
    MaxPooling2D(),
    Conv2D(32, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Conv2D(64, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(1)
])
model.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
model.summary()
```

h. We will use the **fit_generator** method of the **ImageDataGenerator** class to train the network. <u>This step may take a long time.</u>

```
history = model.fit_generator(
    train_data_gen,
    steps_per_epoch=total_train // batch_size,
    epochs=epochs,
    validation_data=val_data_gen,
    validation_steps=total_val // batch_size
)
```

i. Lastly, visualize the accuracy and loss for training and validation.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss=history.history['loss']
val_loss=history.history['val_loss']
epochs_range = range(epochs)
plt.figure(figsize=(8, 8))

plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

4. In a Markdown cell, answer:

   Q1. How would you describe the trend of (1) training accuracy, (2) validation accuracy, (3) training loss, and (4) validation loss?

   Q2. Do you observe any issue in the plots, for example, overfitting?

5. Next, let's try to use regularization techniques, for example, data augmentation. Let's apply 45 degree rotation, width shift, height shift, horizontal flip and zoom (up to 50%) to the training images. We will take one sample image from the training examples and apply transformations to the same image five times. Plot the transformed images. Generally, we only apply data augmentation to the training examples. Therefore, there is no extra transformation applied to validation examples.

```
image_gen_train = ImageDataGenerator(
                    rescale=1./255,
                    rotation_range=45,
                    width_shift_range=.15,
                    height_shift_range=.15,
                    horizontal_flip=True,
                    zoom_range=0.5
                    )
train_data_gen = image_gen_train.flow_from_directory(batch_size=batch_size,
directory=train_dir, shuffle=True, target_size=(IMG_HEIGHT, IMG_WIDTH),
class_mode='binary')

augmented_images = [train_data_gen[0][0][0] for i in range(5)]
plotImages(augmented_images)

image_gen_val = ImageDataGenerator(rescale=1./255)
val_data_gen = image_gen_val.flow_from_directory(batch_size=batch_size,
                                                 directory=validation_dir,
                                                 target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                 class_mode='binary')
```

This is the end of lab 5.

Assignment: continue with your lab 5 notebook, and apply **dropout** to the **first** and **last** max pool layers in step 4-g. Your applied dropout should randomly set **20%** of the neurons to zero during each training epoch. Then repeat the rest of step 4-g (model compile), 4-h (model fit), and 4-i (result visualization) for the new model. In a Markdown cell, answer:

Q3. How would you describe the trend of (1) training accuracy, (2) validation accuracy, (3) training loss, and (4) validation loss?

Q4. Do you think the issue before regularization is solved after regularization?

Hint: you may refer to lab 4 for adding dropout layer with a dropout probability.

Submission:
Finish all the steps in the lab and assignment, save it to lab_assignment5.ipynb file with all the outputs and your answers to the four questions. And submit on Blackboard via submission link.
Due: March 1st 11:30am EST

Reference:
TensorFlow Tutorials: https://www.tensorflow.org/tutorials
https://towardsdatascience.com/exploring-image-data-augmentation-with-keras-and-tensorflow-a8162d89b844