CSC 487: Deep Learning

Homework 2: Facial Keypoint Detection

In this homework you will develop a convolutional neural network (CNN) to detect 15 facial keypoints in an image of a face.

The provided starter notebook shows how to download and unpack the "image" and "keypoints" arrays.

The keypoints are stored as x/y pairs as follows:

```
left_eye_center_x left_eye_center_y
right_eye_center_x right_eye_center_y
left_eye_inner_corner_x left_eye_inner_corner_y
left_eye_outer_corner_x left_eye_outer_corner_y
right_eye_inner_corner_x right_eye_inner_corner_y
right_eye_outer_corner_x right_eye_outer_corner_y
left_eyebrow_inner_end_x left_eyebrow_inner_end_y
left_eyebrow_outer_end_x left_eyebrow_outer_end_y
right_eyebrow_inner_end_x right_eyebrow_inner_end_y
right_eyebrow_outer_end_x right_eyebrow_outer_end_y
nose_tip_x nose_tip_y
mouth_left_corner_x mouth_left_corner_y
mouth_right_corner_x mouth_right_corner_y
mouth_center_top_lip_x mouth_center_top_lip_y
mouth_center_bottom_lip_x mouth_center_bottom_lip_y
```

Be aware that the keypoints contain missing values which are encoded as NaN values (np.nan).

Instructions

1. **Inspect the data.** Print out the shape, dtype, min, and max of the arrays and explain them in your report. Show some of the images and plot the keypoint locations on top of the images.
   Note: You can use np.nanmin() and np.nanmax() to compute the min and max while ignoreing the missing values.

2. **Preprocess and split the data.** Based on your analysis of the data, apply data preprocessing as appropriate. Prepare a 90/10 train/test split.
   Note: You need to be careful when preprocessing the data to avoid the missing values. The scikit-learn preprocessing classes like MinMaxScaler will properly avoid the NaN values so you are recommended to use those.
3. **Prepare Dataset and DataLoader objects.** You can use TensorDataset as in HW1 to load the data.
4. **Create a CNN.** The design of the CNN is up to you. You are welcome to use the "VGG-style" pattern shown in class (several blocks of conv-conv-pool, ending with flatten and finally a linear transformation or MLP).
5. **Train the model.** Train your CNN on the data.
   Note: The CNN will be extremely slow unless you use a GPU (like on Colab). You will need to do .cuda() on your data tensors as well as the model in order to move everything to the GPU.
   Note: You will need to create a custom loss function to avoid the missing values. Here is an example:

```
def masked_mse_loss(y_pred,y_true):
    mask = 1-torch.isnan(y_true).float()
    diff = (y-z)**2
    return (diff*mask).sum()/mask.sum()
```

6. **Analyze the results.** Show some test images and predicted keypoints versus ground truth. Diagnose your initial results in terms of bias and variance, overfitting and underfitting. Be sure to report error metrics in pixel units – for example, if you shifted and scaled the keypoints, or used a squared error loss term, you need to compensate for that when reporting the results to make sure the units are correct.
7. **Improve the model.** Now try at least two different modifications to improve your model. One of the modifications should be some form of data augmentation (see this page). Keep in mind that any geometric augmentations (like crop, rotate, scale) need to act on both the images and the keypoints – Albumentations makes this easy so I recommend using that package.

Report

Your report should include the following:

- **Code explanation:**

    - Briefly explain your solution and any design choices you made that weren't specified in the instructions.
    - Clearly describe any external sources that were used (e.g. websites or AI tools) and how they were used.
- **Discussion:**
    - See questions in step 1.
    - Compare your original and improved models in terms of test performance and overfitting.

Deliverables

Python notebook and report document due Sunday, Feb. 16, 11:59 pm.