Coding assigment on CNNs and Transfer learning

The goal of this assignment is to train a Convolutional Neural Network (CNN) on the Standford dog breed dataset and compare its performance to a Transfer Learning approach.

Dataset

We will be using the Stanford Dogs dataset which consists of 20,580 images of 120 dog breeds. You can download the dataset here.

- Split the dataset into train and validation set with **80% for training and 20% for validation**

Task

Your task is to create two convolutional neural network (CNN) models to classify the images of dog breeds: one using normal CNN and the other using transfer learning. You will compare the performance of the two models on the test set

Additional tasks

- Experiment with hyperparameters
- Avoid overfitting

Steps

Load and preprocess the dataset:

- Load the images into memory using `ImageDataGenerator`. Refer to this notebook if you do not know how to load image data
- Refer to this notebook on how to load data using `ImageDataGenerator`
- Resize the images to a standard size (e.g., 224 x 224).
- Normalize the pixel values of the images to be between 0 and 1. (rescale)
- Split the dataset into training and validation sets.

Create a normal CNN model:

- Create a sequential model with multiple convolutional and pooling layers.
- Add a flatten layer to convert the output of the convolutional layers to a 1D vector.
- Add a fully connected layer with the appropriate number of neurons for the output classes (in our case, 120).
- Add a softmax activation function to the output layer to convert the outputs to probabilities.

Train the normal CNN model:

- Compile the model with an appropriate loss function, optimizer, and evaluation metric.
- Train the model on the training set using the fit() function.
- Monitor the validation loss and accuracy during training to avoid overfitting.

Create a transfer-learning/fine-tuning CNN model:

- Load a pre-trained CNN model (e.g., VGG16, ResNet50, etc)
  - You can find the pretrained models for tensorflow in their library at `tf.keras.applications`
  - Go through the docs and understand how to use it
- You can either freeze the entire pre-trained model or fine-tune it by freezing top-n layers.
- Add custom layers on top of the pre-trained model:
  - Add a global average pooling layer to reduce the dimensionality of the output.
  - Add a fully connected layer with the appropriate number of neurons for the output classes (in our case, 120).

    - Add a softmax activation function to the output layer to convert the outputs to probabilities

Train the transfer learning CNN model:

- Compile the model with an appropriate loss function, optimizer, and evaluation metric.
- Train the model on the training set using the fit() function.
- Monitor the validation loss and accuracy during training to avoid overfitting.

Important

- While using the `fit` function, store the output in a variable

```
history1 = model1.fit(...)
```

- You can use data augmentation
- You can use callbacks to control the training

Evaluate the models and save results

- Compute the `accuracy, precision, recall, and F1-score` of both models on the validation set. (use `sklearn`)
- Create a dataframe with the columns: `accuracy, precision, recall, and f1-score`, model
    - model = 'Normal' for regular CNN
    - model = 'Pretrained' for Pretrained model
- Save the CSV file as `SRN_A1_TDL23.csv`

Save the history of both the models as follows (Please follow the naming convention as shown)

```
import pickle
with open("SRN_history_Normal.pkl", "wb") as f:
    pickle.dump(history1.history,f)
```

```
import pickle
with open("SRN_history_Pretrained.pkl", "wb") as f:
    pickle.dump(history2.history,f)
```

- **Note that the `fit` function will return None if it is interrupted, so make sure to finish the training completely**

Evaluation Criteria

Your submission will be evaluated on the following criteria:

Code Quality:

- Is the code well-structured and readable?
- Are the variable and function names descriptive and meaningful?
- Are the comments clear and concise?

Functionality:

- Does the code load and preprocess the dataset correctly?
- Does the code create and train the normal CNN model correctly?
- Does the code create and train the transfer learning CNN model correctly?
- Does the code evaluate the models and compute their performance metrics correctly?

Model Performance:

- Is your model overfitting? (Important)
- What is the accuracy, precision, recall, and F1-score of the normal CNN model on the validation set?
- What is the accuracy, precision, recall, and F1-score of the transfer learning CNN model on the validation set?
- How much improvement in performance does the transfer learning model provide compared to the normal CNN model?

How to submit

- Zip your into `SRN_A1_TDL23.zip`
  - Notebook; Named as `SRN_A1_TDL23.ipynb`
  - CSV file; Named as `SRN_A1_TDL23.csv`
  - Pickles files: Named as `SRN_history_Normal.pkl` and `SRN_history_Pretrained`
- Google form will be provided to submit

Good luck!