Exercise 2

Title: Image Recognition to Identify Species of Flowers

Abstract:

This study explores image classification using Convolutional Neural Networks (CNN) on the TensorFlow flower dataset. The dataset comprised 3670 samples and 5 classes of flowers to be classified namely daisies, dandelions, roses, sunflowers, and tulips with varying numbers of image samples each. Downloaded directly from the TendorFlow website, the data underwent a series of experiments including regularization methods like data augmentation (random flip and random rotation), transfer learning methods like VGG16, dropout rates, early stopping, and hyperparameter tuning. As a result, training and validation accuracies of 0.95 and 0.85 respectively were achieved.

Introduction:

"Unlike the standard neural network consisting of fully connected layers only, CNN consists of at least one convolutional layer." (Kim et al., 2022:2-3). Here the word convolution reflects the entwining of information before feeding it into the model depicting the human neurons. A transfer learning method, VGG16 has previously displayed robust performance for image classification tasks. This type of learning comes in handy when we have a pre-trained model like ours, that has already been trained on a large dataset, and used as a fixed feature extractor. Unfreezing some of the layers of the pre-trained model and retraining them along with the newly added layers. This can help adapt the pre-trained model more closely to the specific characteristics of the new task.

In our model, the feature extraction approach of transfer learning was employed, as there are two to choose from:

> "Roughly, there are two TL approaches to leveraging CNN models: either feature extractor or fine-tuning. The feature extractor approach freezes the convolutional layers, whereas the fine-tuning approach updates parameters during model fitting." (Kim et al., 2022:4).

Materials:

Python and Google Colab for such deep learning tasks are always the preferred tools as it provides efficiency, especially for neural networks having multiple layers and parameters that impact memory usage during training. These models also require larger memory and storage spaces to function properly which is again a facility found in Google Colab. The libraries used were namely TensorFlow, Matplotlib, and tarfile.

Methodology:

The methodology followed was as straightforward as training a CNN model for image classification using the best set of preprocessing techniques, hyperparameters, and regularization methods. The reason why CNN, today, is so widely used and is the preferred model for image classification tasks is that:

> “Compared with traditional machine learning, CNN has more advantages, on the one hand, it has more hidden layers and complex network structure, and on the other hand, it has a stronger ability of feature learning and feature expression.” (Zhao Jiantao and Chu Shumin, 2021:1).

Data Extraction:

The data was downloaded directly from the Tensorflow site in a .tgz extension file, placed in a separate path in the directory, and then extracted into the environment.

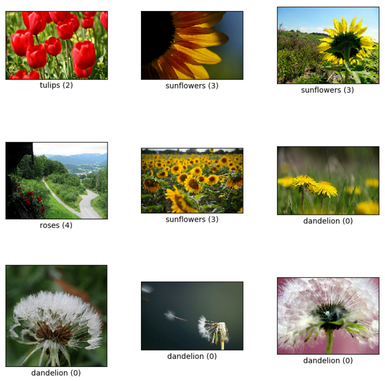

Fig 1. Flower Sample Images from tf_flowers dataset (tensorflow.org, 2024)

Hyperparameter Selection:

The hyperparameters for training and validating the data were set initially to ensure that the experiments were consistent and reproducible. Below is the depiction of the values chosen for each hyperparameter:

IMAGE_SIZE = 256

BATCH_SIZE = 32

CHANNELS = 3

EPOCHS = 50

We then applied a data augmentation pipeline technique using Keras's Sequential API. This helped the model learn each sample from different perspectives i.e. by applying random flipping and rotation methods, thereby improving the training process of the model by creating variants (instances) of each sample (image).

Data Preprocessing:

The next step was to set up the preprocess_input function from keras to our images. This feature normalizes the pixel values of the input images before feeding them into the VGG16 model. It also helps in ensuring that the images are preprocessed in a manner consistent with how the model was originally trained i.e. by color channel adjustment, and resizing as required by the model's architecture. This architecture helps in enhancing the preprocessing of image data and most commonly for image classification and object recognition tasks. It mainly is comprised “of 16 layers, including 13 convolutional layers and 3 fully connected layers.” (GeeksforGeeks, 2024)

Data Augmentation:

We then made use of ImageDataGenerator; a feature provided by TensorFlow Keras to generate batches of image data with real-time data augmentation, which was initiated earlier. This step increases the model’s ability to generalize to new or unseen data and diversify the transformed images to improve the learning rate. This is how it works:

> “The generator will run through your image data and apply random transformations to each individual image as it is passed to the model so that it never sees the exact same image twice during training.” (Brandon Walraven, 2019)

This step would help in loading and transforming the images on the fly during training. We spared 2939 images belonging to 5 classes for training and 731 images belonging to 5 classes for validation.

Model Building:

The CNN model was build using the VGG16 architecture; a type of transfer learning model, as the base model (which was configured earlier). The VGG16 model is pre-trained on ImageNet and is set to non-trainable. Additional layers are added for data augmentation, global average pooling, dense layer with ReLU activation, dropout for regularization in between these layers, and a final dense (output) layer using softmax activation for classification.

Regularization:

The dropout rate was set to 0.5 as this rate aids the model with generalizing better by ensuring it does not become too reliant on specific neurons.

Model Compiling:

The model itself was then compiled using the optimizer set as adam, the loss function set as categorical cross-entropy, and the evaluation metric accuracy.

```
Layer (type)                 ┃ Output Shape          ┃     Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┩
│ sequential (Sequential)      │ (None, 256, 256, 3)   │           0 │
├──────────────────────────────────────────────────────────────┼─────────────────
────────┼────────────────────────┤
│ vgg16 (Functional)           │ (None, 8, 8, 512)     │  14,714,688 │
├──────────────────────────────────────────────────────────────┼─────────────────
────────┼────────────────────────┤
│ global_average_pooling2d_1   │ (None, 512)           │           0 │
│ (GlobalAveragePooling2D)     │                       │             │
├──────────────────────────────────────────────────────────────┼─────────────────
────────┼────────────────────────┤
│ dense_2 (Dense)              │ (None, 1024)          │     525,312 │
```

```
├──────────────────────────────────────┼──────────────────────
──────┼─────────────────┤
│ dropout_1 (Dropout)          │ (None, 1024)          │          0 │
├──────────────────────────────────────┼──────────────────────
──────┼─────────────────┤
│ dense_3 (Dense)          │ (None, 5)          │      5,125 │
└──────────────────────────────────────┴──────────────────────
──────┴─────────────────┘
```

Table 1. Depiction of the model summary.

The above summary reflects the parameters included in the VGG16 model including the weights and biases of all the convolutional and fully connected layersas its a complex, deep network, it has a large number of parameters i.e. Then, the global average pooling layer is depicted the way it calculates the average of each feature map of the VGG16 output. FInally, the dense layers, with their respective number of neurons, weights and biases with a dropout layer in between in summarized.

Callback Functions:

Approaches like Early stopping and model checkpointing were utilized to stop training when validation loss does not improve for 10 consecutive epochs, and checkpointing then saves the model with the best validation loss. This resulted in epochs stopping at 17 when the model validation loss stopped improving.

Results:

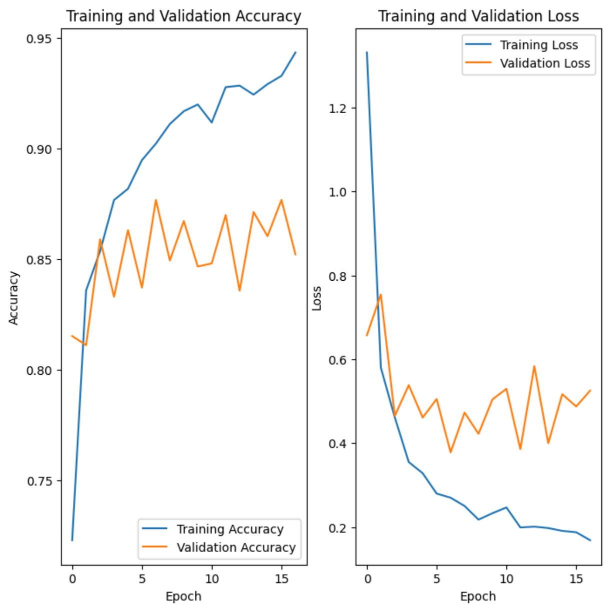

Fig 2. Figure depicts both the training and validation accuracy and loss over 15 epochs for our CNN model.

As a result, training and validation accuracies of 0.95 and 0.87 respectively were achieved after experimenting several times with regularization methods and hyperparameter tuning.

Discussions:

- The architecture of the CNN model we used incorporates data augmentation to increase the robustness of our model. As mentioned earlier, this method applied random flips and rotations using the ‘ImageDataGenerator’. The pre-trained VGG16 model is used for its feature extraction approach, and the input data is preprocessed accordingly to match VGG16's requirements, ensuring compatibility and leveraging its pre-learned features for improved performance.
- To regularize our model, a droput layer of 0.5, data augmentation techniques like image flipping set to horizontal_and_vertical and rotation set to 0.2 were applied. This forces the model to generalize better.
- The model exhibited overfitting which was brought to balance by tuning number of layers and units ensuring that the model has enough capacity to learn from the data without overfitting.

- The model indicated some overfitting so methods like early stopping and learning rates were used to improve the validation loss.

References:

Kim, H.E., Cosa-Linan, A., Santhanam, N. et al. Transfer learning for medical image classification: a literature review. BMC Med Imaging 22, 69 (2022). https://doi.org/10.1186/s12880-022-00793-7

Jiantao, Zhao & Shumin, Chu. (2021). Research on Flower Image Classification Algorithm Based on Convolutional Neural Network. Journal of Physics: Conference Series. 1994. 012034. 10.1088/1742-6596/1994/1/012034.

Tensorflow (2024) tf_flowers , [Photograph]. Available online: https://www.tensorflow.org/datasets/catalog/tf_flowers [Accessed 17/6/2024].

GeeksforGeeks (2024), VGG-16 | CNN model, Available online: https://www.geeksforgeeks.org/vgg-16-cnn-model/ [Accessed 20/6/2024].

Brandon Walraven (2019), Boost Your CNN with the Keras ImageDataGenerator. Medium. Available online: https://medium.com/@bcwalraven/boost-your-cnn-with-the-keras-imagedatagenerator-99b1ef262f47#:~:text=The%20ImageDataGenerator%20is%20a%20class,other%20object%20in%20the%20library.&text=The%20generator%20will%20run%20through,same%20image%20twice%20during%20training. [Accessed 21/6/2024]