Data Mining Homework 2

Option2: Image classification based on Deep Learning

classify flower dataset and compare accuracy of VGG and inceptionv3 model.

Github Link : https://github.com/vaibhavisavani1910/Flower-dataset-classification

Name: Vaibhavi Hiteshkumar Savani
SJSU ID: 017456972

Professor: Kaikai Liu

Use Classification techniques on flower dataset to classify each Image of the flower. Experiment with different machine learning models and compare their training accuracy and performance.

Step 1 : Download flower dataset

Utilized Kaikai Liu's code to download the flower dataset from the Google API. Used the code from MultiModalClassifier to download images to the HPC cluster. There were a total of 3670 images and 5 classes of flowers - dandelion, roses, tulips, sunflowers, and daisies. The dataset had 5 folders with the names of the types of flowers and contained the respective flower images.

Step 2 : Data pre-processing

Utilized the Keras library's ImageDataGenerator class to implement real-time data augmentation on image data. Applied various transformations to the input images to artificially increase the size of the training dataset and enhance the model's generalization. Resized the images to a size of 224, which is suitable for our model.

Step 3 : Train Test split

The dataset consisted of five folders, each containing images of a single class. To split the dataset into training and testing sets, I saved the list of images and class information into a dataframe. The dataset was divided, with 20% allocated for the test dataset.

Step 4 : Training with Custom VGG19 model

I utilized the VGG19 model to train images from the training dataset. VGG-19 is a convolutional neural network consisting of 19 layers. Utilizing a pre-trained model with ImageNet weights, I extended the architecture with an additional deep network stacked on top of VGG19. The output of the VGG19 model was flattened before being fed into fully connected layers. Two Dense layers with 4096 units each and ReLU activation functions were incorporated, followed by dropout layers with a rate of 0.5. Dropouts aid in preventing overfitting by randomly setting a fraction of input units to zero during training. Finally, a Dense layer with 5 units and a softmax activation function was added as the output layer, representing the classification of 5 classes.

```python
# Load the VGG19 model without the top layers
vgg19 = VGG19(include_top=False, weights='imagenet', input_shape=(224, 224, 3))

# Freeze the weights of the VGG19 layers
for layer in vgg19.layers:
    layer.trainable = False

# Build the custom model
vggModel = Sequential([
    vgg19,
    layers.Flatten(),
    layers.Dense(4096, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(4096, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(5, activation='softmax'),
])

# Compile the model
vggModel.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Figure 1: Custom VGG19 model

```
Layer (type)                 Output Shape              Param #
=================================================================
vgg19 (Functional)           (None, 7, 7, 512)         20024384

flatten (Flatten)            (None, 25088)             0

dense (Dense)                (None, 4096)              102764544

dropout (Dropout)            (None, 4096)              0

dense_1 (Dense)              (None, 4096)              16781312

dropout_1 (Dropout)          (None, 4096)              0

dense_2 (Dense)              (None, 5)                 20485

=================================================================
Total params: 139,590,725
Trainable params: 119,566,341
Non-trainable params: 20,024,384
_________________________________________________________________
```

Figure 2: Custom VGG19 model

"The next step involved training our custom VGG19 model using the flower dataset. I leveraged the HPC cluster to expedite the training process and achieve faster results. Following the training of our custom VGG19 model, I attained an accuracy of 75%.

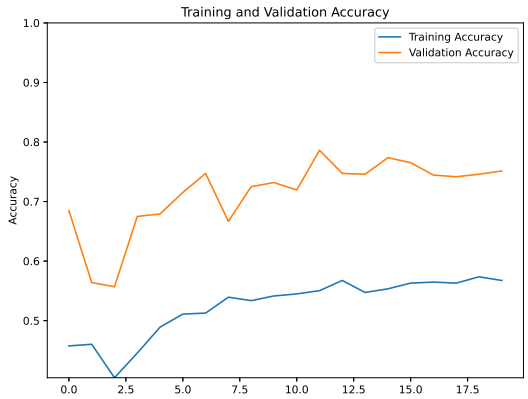

Figure 3: Trainng and Validation Accuracy for VGG19 Model

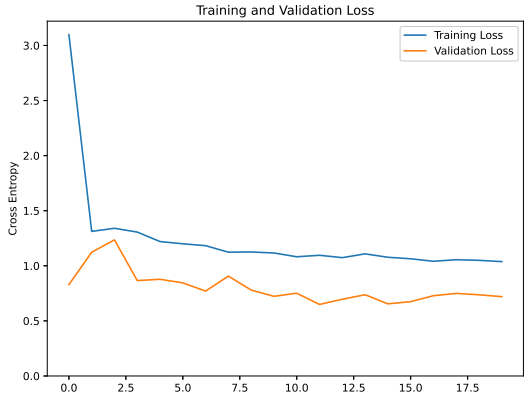

Figure 4: Trainng and Validation Loss for VGG19 Model

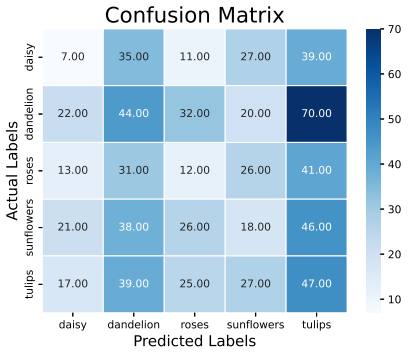

Figure 5: Confusion Matrix for VGG19 Model

Step 5 : Train model with InceptionV3

InceptionV3 is a deep convolutional neural network with 48 layers. I constructed a custom model on top of the InceptionV3 architecture. The GlobalAveragePooling2D() function was employed to reduce the spatial dimensions of the output, providing a global average over each feature map. This aids in diminishing the number of parameters and computational complexity.

```python
# Load the InceptionV3 model without the top layers
inception = InceptionV3(include_top=False, weights='imagenet', input_shape=(224, 224, 3))

# Freeze the weights of the InceptionV3 layers
for layer in inception.layers:
    layer.trainable = False

# Build the custom model
incModel = Sequential([
    inception,
    layers.GlobalAveragePooling2D(),
    layers.Dense(1024, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(5, activation='softmax'),
])

# Compile the model
incModel.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Print the model summary
incModel.summary()
```

Figure 6: InceptionV3 Custom Model

The custom model includes a dense layer with 1024 units and a ReLU activation function. To prevent overfitting, a dropout layer with a rate of 0.5 was added. Finally, a dense layer with 5 units and a softmax activation function was included as the output layer, representing the classification of 5 classes.

In summary, this code defines a custom model for image classification, utilizing the InceptionV3 architecture with pre-trained weights. Additional layers are added on top for fine-tuning on a specific task with 5 classes.

Train model using our training dataset. This custom model gave the accuracy of 88%.

```
Downloading data from https://storage.googleapis.com/tensorflow/ker
87910968/87910968 [==============================] - 1s 0us/step
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_v3 (Functional)   (None, 5, 5, 2048)        21802784

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dense_3 (Dense)             (None, 1024)              2098176

 dropout_2 (Dropout)         (None, 1024)              0

 dense_4 (Dense)             (None, 5)                 5125

=================================================================
Total params: 23,906,085
Trainable params: 2,103,301
Non-trainable params: 21,802,784
_________________________________________________________________
```

Figure 7: InceptionV3 Custom Model

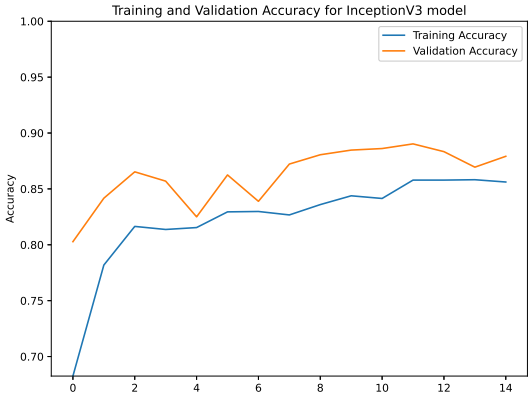

Figure 8: Trainng and Validation Accuracy for InceptionV3 Model

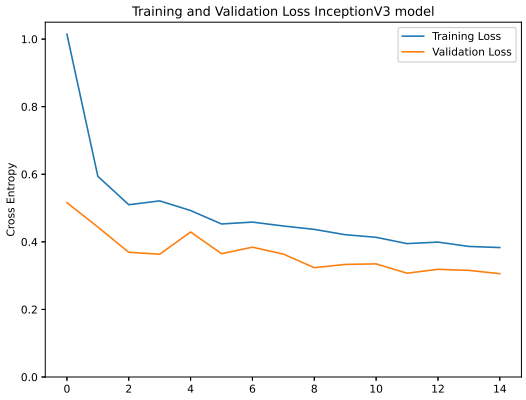

Figure 9: Trainng and Validation Loss for InceptionV3 Model

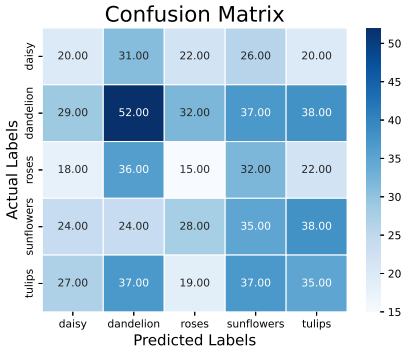

Figure 10: Confusion Matrix for InceptionV3 Model

Step 6: Save the best model

After analyzing both models, we will save the model that demonstrated superior performance. The Custom InceptionV3 model gave the best results; therefore, we will save it along with its weights for future use, deployment, or further analysis. This ensures that the trained model's architecture, parameters, and learned patterns are preserved, allowing for seamless integration into applications or additional evaluations.

Step 7: Predict

Lastly, to analyze where our model is providing good results and where it is not, we will examine the results of predictions where our algorithm performed well and where our model did not perform well.