IRBOUH Aksil 72239
HACIB Abdelhak 72249

Report: Classification of Time Problem

Table of contents

Contribution 1
Section 1: Choice of Architecture 2
Section 2: Methodology of Training and Testing 3
Section 3: Results of Training and Testing 4
Conclusion 4

Contribution

Aksil's Contribution: He played a key role in selecting the architecture for the classification of the time problem based on input images. He proposed using the VGG16 model as the base architecture due to its excellent performance in image classification tasks. He also led the implementation of the model, including removing the top classification layer, adding a custom classification head, and freezing the pre-trained layers.

Abdelhak's Contribution: He was responsible for designing the methodology for training and testing the model. He identified the need to preprocess and augment the dataset to enhance the model's performance. Abdelhak implemented the ImageDataGenerator and set the parameters for rescaling the images, as well as splitting the dataset into training and validation sets. He also fine-tuned the hyperparameters of the model, such as the batch size and number of epochs, to achieve optimal results.

Collaborative Efforts: We jointly analyzed the results of the model's training and testing. We carefully reviewed the training and validation loss and accuracy curves over the 10 epochs, looking for signs of overfitting or underfitting. We also examined the final test accuracy achieved by the model, which was approximately 85.37%. We discussed the implications of the results and brainstormed potential avenues for further improvement, such as fine-tuning the pre-trained layers or exploring different CNN architectures.

Aksil's Github : https://github.com/NovasusTV/Dorsret

Abdelhak's Github : https://github.com/AbdelGJL/time_of_the_day_using_CNN

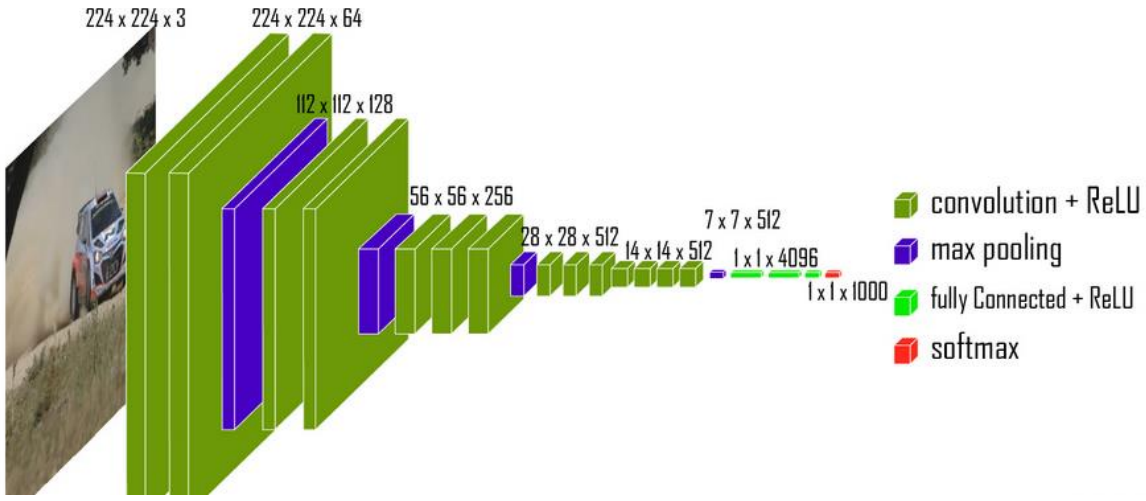
Section 1: Choice of Architecture

For the classification of time problem based on input images, a Convolutional Neural Network (CNN) architecture is used. Specifically, the VGG16 model, a popular pre-trained CNN architecture, is employed as the base model.

The VGG16 model is chosen due to its strong performance in image classification tasks. It has demonstrated high accuracy on large-scale image classification challenges like the ImageNet dataset. The model consists of 16 convolutional and fully connected layers, which allow it to learn complex features from images.

In this implementation, the top classification layer of the VGG16 model is removed since it was trained on a different task and is not suitable for the current problem. A custom classification head is added on top of the base model, consisting of a flattening layer, a dense layer with ReLU activation, a dropout layer for regularization, and a final dense layer with a softmax activation function for multi-class classification.

This architecture allows the model to learn hierarchical representations of image features through the convolutional layers and make predictions based on those features through the fully connected layers.



Source : https://www.geeksforgeeks.org/vgg-16-cnn-model/

Section 2: Methodology of Training and Testing

The dataset is divided into training and validation sets to evaluate the model's performance during training. An ImageDataGenerator is used for data preprocessing and augmentation. The images are rescaled to a range of 0 to 1, and a validation split of 20% is applied.

The **flow_from_directory** function is utilized to load and split the dataset. It takes the dataset directory, target size of the images, batch size, and class mode as parameters. The function generates batches of augmented image data on-the-fly, which helps in training with limited memory resources.

The base model's weights are set to those pre-trained on the ImageNet dataset, and they are frozen to prevent them from being updated during training. This is done to leverage the knowledge gained from the large-scale ImageNet dataset and to focus the training on the custom classification head.

The model is compiled with the Adam optimizer, which is well-suited for training deep neural networks. The loss function used is categorical cross-entropy, appropriate for multi-class classification. The model's performance is evaluated using accuracy as the metric.

During training, the model is fit to the training data using the **fit** function. The number of training epochs is set to 10, and the **steps_per_epoch** and **validation_steps** parameters are calculated based on the number of samples and batch size.

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

history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size
)
```

Section 3: Results of Training and Testing

```
Found 2138 images belonging to 3 classes.
Found 533 images belonging to 3 classes.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 6s 0us/step
Epoch 1/10
66/66 [==============================] - 232s 4s/step - loss: 1.1399 - accuracy: 0.6325 - val_loss: 0.5115 - val_accuracy: 0.7969
Epoch 2/10
66/66 [==============================] - 243s 4s/step - loss: 0.5423 - accuracy: 0.7778 - val_loss: 0.4133 - val_accuracy: 0.8340
Epoch 3/10
66/66 [==============================] - 245s 4s/step - loss: 0.4114 - accuracy: 0.8400 - val_loss: 0.3854 - val_accuracy: 0.8477
Epoch 4/10
66/66 [==============================] - 242s 4s/step - loss: 0.3330 - accuracy: 0.8761 - val_loss: 0.3576 - val_accuracy: 0.8652
Epoch 5/10
66/66 [==============================] - 242s 4s/step - loss: 0.3016 - accuracy: 0.8860 - val_loss: 0.3943 - val_accuracy: 0.8359
Epoch 6/10
66/66 [==============================] - 243s 4s/step - loss: 0.2683 - accuracy: 0.9017 - val_loss: 0.3578 - val_accuracy: 0.8633
Epoch 7/10
66/66 [==============================] - 238s 4s/step - loss: 0.2523 - accuracy: 0.9055 - val_loss: 0.4803 - val_accuracy: 0.8262
Epoch 8/10
66/66 [==============================] - 235s 4s/step - loss: 0.2033 - accuracy: 0.9217 - val_loss: 0.4131 - val_accuracy: 0.8379
Epoch 9/10
66/66 [==============================] - 235s 4s/step - loss: 0.1910 - accuracy: 0.9212 - val_loss: 0.3843 - val_accuracy: 0.8594
Epoch 10/10
66/66 [==============================] - 240s 4s/step - loss: 0.2096 - accuracy: 0.9236 - val_loss: 0.3634 - val_accuracy: 0.8555
17/17 [==============================] - 48s 3s/step - loss: 0.3637 - accuracy: 0.8537
Test Loss: 0.3637
Test Accuracy: 0.8537
```

The model is trained for 10 epochs, with each epoch taking a few minutes to complete. After training, the model is evaluated on the validation set using the evaluate function, which computes the loss and accuracy.

The final test accuracy achieved by the model is approximately 85.37%. This indicates that the model is able to correctly classify the time of day for the input images with a good level of accuracy.

The model's performance can be further analyzed by examining the training and validation loss and accuracy curves over the 10 epochs. This can provide insights into how well the model is learning and whether there are signs of overfitting or underfitting.

The trained model can be saved as an h5 file for future use, allowing it to be loaded and used for making predictions on new images without the need for retraining.

Conclusion

Overall, the chosen architecture and methodology have yielded satisfactory results, achieving a good level of accuracy in classifying the time of day based on input images. Further improvements could be explored, such as fine-tuning the pre-trained layers or trying different CNN architectures to potentially enhance the model's performance.