Option 1: Classification of Time Problem

Determine the time of the day based on the input image.
Dataset to be used: https://www.kaggle.com/datasets/aymenkhouja/timeofdaydataset

Link:

Project link (Github/ Gitlab)

https://github.com/ThomasValesi/Data-Group-project

https://colab.research.google.com/drive/1m4RyektQxAx5mPBEnGce4enPXYS9rv0u?usp=sharing

Team Members:

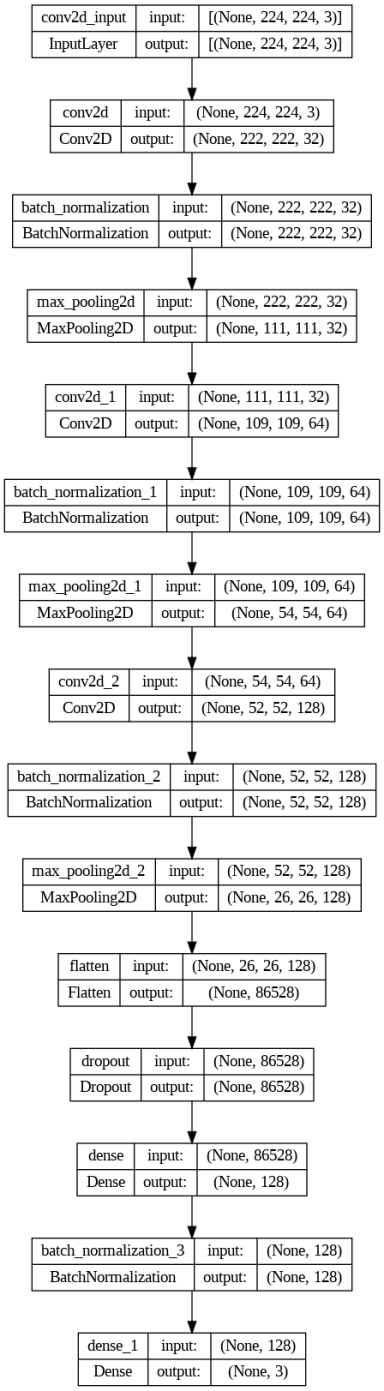
Student Name	Student ID	Contribution in the project
Damien SUAREZ	73116	Fonction to use the model and create all the graphs / Result
Charles PERIER	73101	Function To import dataset and resize images / Data Description
Thomas VALESI	73100	Build the model training and test it / Model Architecture and Methodology

Model Architecture:

Explain the following in this page:

1. What model architecture have you selected?
   We selected the Convolutional Neural Network (CNN) architecture.

2. Put a diagram of the architecture (Schematic)

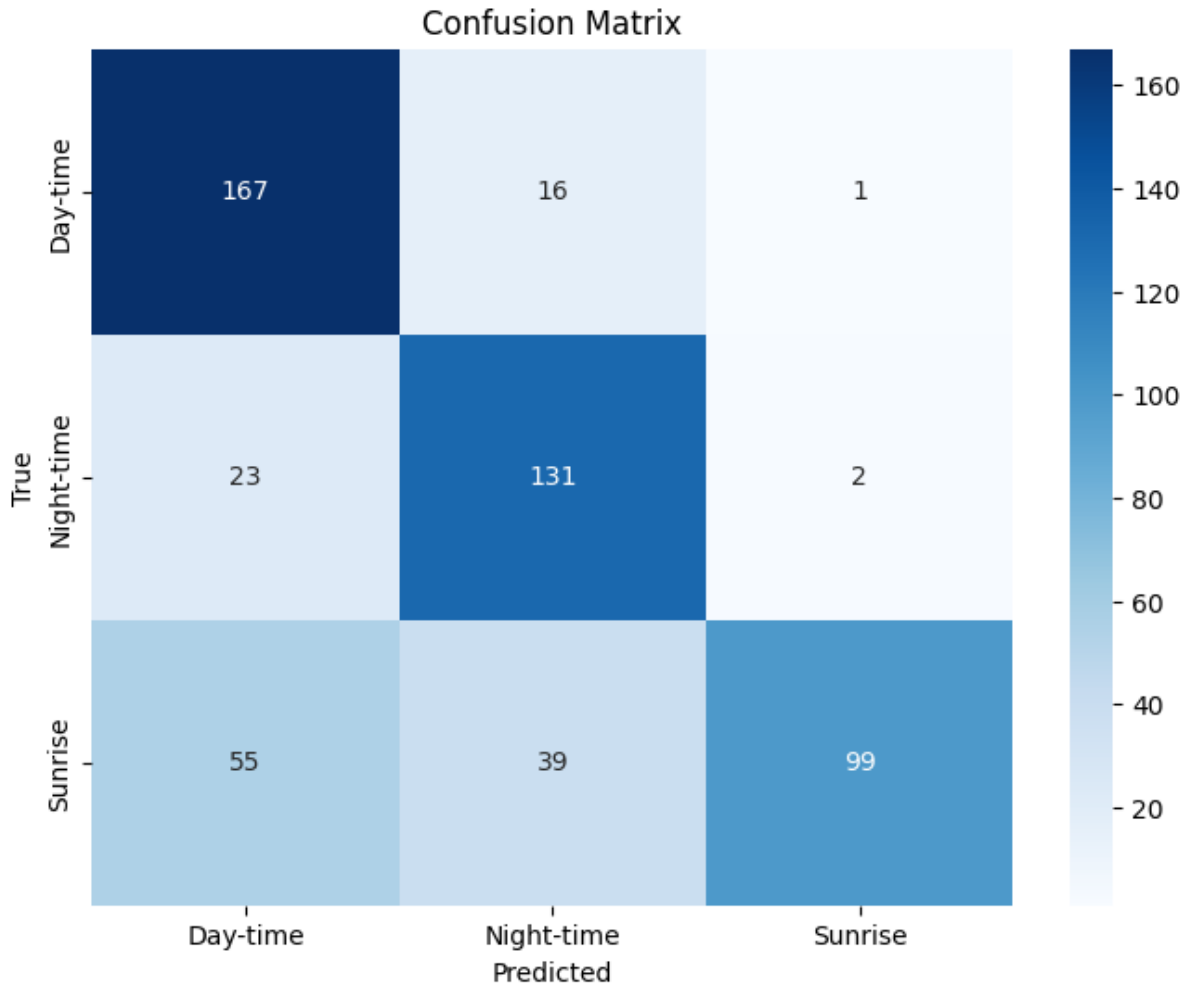


3. How many layers of the model have?
   We have 13 layers of the model.

4. How many hidden layers and neurons do you have?
   In the code there are 2 hidden layers : the Conv2D layer and the dense layer.
   We have 32, 64, 128 neurons in the Conv2D layer and 128 in the dense layer.

5. Have you changed the model architecture to improve performance.
   Yes, we added some step to improve the model architecture, we added some layers and improve some number of neurons.
   We added a program that allows inputting any image and any size and resizes it so that it can work with this model and thus be predicted.

6. Put the confusion matrix :



Dataset Description:

Explain the following in this page:

1. Details of the dataset (number of classes/ instances/ Images etc)
   In this dataset there are 3 classes (Daytime, Night-time and Sunrise)
   The 978 images in Datetime, 762 in Night-Time and 1040 in sunrise.
   There are 2780 images in the dataset. All the classes contain different images of day, night and sunrise, all of them have the same size (224x224 pixels).

2. What kind train / test split has been used
   We use 80% of train and 20% of test.

3. What kind of data augmentation have been used?
   We used data augmentation, which generates new images from the dataset. In our case, it allows the creation of images by zooming, flipping, and resizing.

Methodology:

Explain the following in this page:

1. Training parameters, number of epochs, learning rate etc.
   The model is trained using the Adam optimizer with a default learning rate. The training process is carried out for 10 epochs, and the batch size is determined implicitly by the ImageDataGenerator. Data augmentation parameters, including rotation, width shift, height shift, shear, zoom, and horizontal flip, are specified in the ImageDataGenerator.

2. Loss function used for training
   The chosen loss function is sparse categorical crossentropy (loss='sparse_categorical_crossentropy' in model.compile). The evaluation metric during training is accuracy (metrics=['accuracy'] in model.compile).

3. Changes is parameter affecting the model results :
   The image generator improved the model result. We went from 40% to nearly 80% accuracy with this. The change in the number of epochs, as well as the model itself, also affects the results of the model.

4. How have you improved the results?
   With Image Generator and with the improve of number of layers in the CNN model.

Results:

Explain the following in this page:

1. Results of training (loss and accuracy of the model)

```
Epoch 1/10
67/67 [==============================] - 410s 6s/step - loss: 0.6572 -
accuracy: 0.7989 - val_loss: 1.2513 - val_accuracy: 0.7373
Epoch 2/10
67/67 [==============================] - 397s 6s/step - loss: 0.4480 -
accuracy: 0.8322 - val_loss: 0.7063 - val_accuracy: 0.7692
Epoch 3/10
67/67 [==============================] - 397s 6s/step - loss: 0.3980 -
accuracy: 0.8492 - val_loss: 0.5512 - val_accuracy: 0.8424
Epoch 4/10
67/67 [==============================] - 378s 6s/step - loss: 0.3920 -
accuracy: 0.8473 - val_loss: 0.6851 - val_accuracy: 0.7899
Epoch 5/10
67/67 [==============================] - 390s 6s/step - loss: 0.3639 -
accuracy: 0.8595 - val_loss: 0.5059 - val_accuracy: 0.8311
Epoch 6/10
67/67 [==============================] - 397s 6s/step - loss: 0.3268 -
accuracy: 0.8665 - val_loss: 0.5269 - val_accuracy: 0.8049
Epoch 7/10
67/67 [==============================] - 395s 6s/step - loss: 0.3466 -
accuracy: 0.8665 - val_loss: 0.4198 - val_accuracy: 0.8574
Epoch 8/10
67/67 [==============================] - 392s 6s/step - loss: 0.3454 -
accuracy: 0.8637 - val_loss: 0.5811 - val_accuracy: 0.7899
Epoch 9/10
67/67 [==============================] - 398s 6s/step - loss: 0.3298 -
accuracy: 0.8731 - val_loss: 0.3794 - val_accuracy: 0.8799
Epoch 10/10
      67/67 [==============================] - 406s 6s/step - loss:
      0.3263 - accuracy: 0.8863 - val_loss: 0.4943 - val_accuracy:
      0.8368
```

2. Results on testing (loss and accuracy of the model)

```
17/17 [==============================] - 23s 1s/step - loss: 0.4943 -
accuracy: 0.8368
      Test Accuracy: 0.8367729783058167
```

3. Show some direct results: example input picture/ sentence and output results.

Night-Time


Day-Time


Sunrise