Gliwice 29.06.2024r.

Report

Biologically Inspired Artificial Intelligence

Theme:

Weather recognition

Author:

Paweł Ledwoń

1. Aim of the project
   The aim of the project was to prepare an application which would be able to recognize weather presented on image.

2. Technology
   2.1. Programming language
   Python was chosen to prepare such application. The reason was the ease of use for machine learning and data processing tasks.

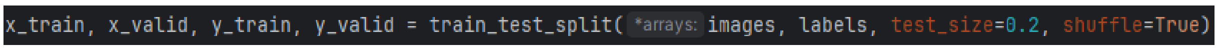
   2.2. Libraries
   2.2.1. TensorFlow
   Used for building and training the neural network model. TensorFlow offers huge support for deep learning architectures.
   2.2.2. OpenCV
   Used for image processing tasks such as resizing and augmenting images, ensuring that images properties are similar.
   2.2.3. Scikit-Learn
   Provided tools for splitting the dataset into training and validation sets, as well as utilities for presenting model performance.
   2.2.4. NumPy
   Essential for handling arrays and numerical data, enabling efficient manipulation and preprocessing of the image data.

   2.3. Hardware
   Training the model was accelerated using a GPU, significantly reducing the training time and enabling using a larger dataset. It reduced the training time about 10 times.

3. Dataset
   3.1. The dataset consists of 11 different weather conditions:
   dew, fogsmog, frost, glaze, hail, lightning, rain, rainbow, rime, sandstorm, snow

   3.2. Each image in the dataset was resized to 180x180 pixels to ensure uniform input dimensions for the neural network.

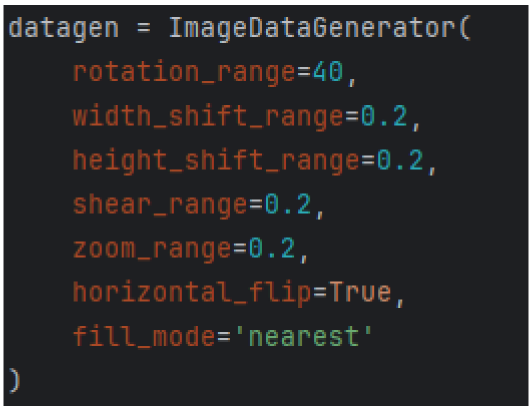
   3.3. It was split into training and validations sets, with 20% of the data allocated for validation to ensure model's best performance.



```
x_train, x_valid, y_train, y_valid = train_test_split( *arrays: images, labels, test_size=0.2, shuffle=True)
```

Image 1. Train – test split

   3.4. To enhance the model's generalization capabilities, data augmentation techniques were applied. These included rotations, shifts, shearing, zooming, and horizontal flipping. This process artificially increased the diversity of the training data, helping the model perform better.



```
datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

Image 2. Data augmentation

4. Model

   The model was built using TensorFlow's Sequential API, structured as follow:

   4.1. Convolutional Layers

   Four convolutional layers with ReLU activation and varying filter sizes were used to extract features from the images. The kernel regularizer (l2) was applied to the first layer to prevent overfitting.

   4.2. MaxPooling Layers

   Each convolutional layer was followed by a MaxPooling layer to reduce the spatial dimensions of the feature maps and retain the most significant features.

   4.3. Flatten Layer

   The final convolutional output was flattened into a one-dimensional vector.

   4.4. Dense Layer

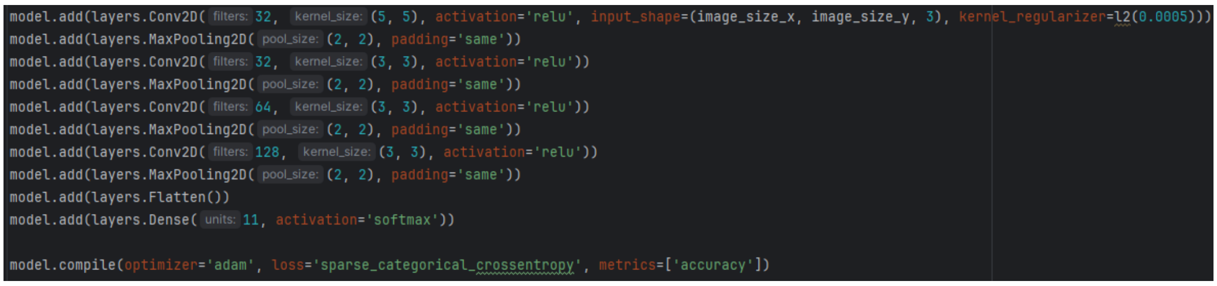
   A dense layer with a softmax activation function was used to produce the final output, representing the probability distribution over the 11 weather condition classes.

   4.5. Compilation

   The model was compiled with the Adam optimizer, chosen for its efficiency and adaptive learning rate capabilities.

   4.6. Training

   The model was trained using the augmented data over 100 epochs, with a batch size of 32.



```
model.add(layers.Conv2D( filters: 32,  kernel_size: (5, 5), activation='relu', input_shape=(image_size_x, image_size_y, 3), kernel_regularizer=l2(0.0005)))
model.add(layers.MaxPooling2D( pool_size: (2, 2), padding='same'))
model.add(layers.Conv2D( filters: 32,  kernel_size: (3, 3), activation='relu'))
model.add(layers.MaxPooling2D( pool_size: (2, 2), padding='same'))
model.add(layers.Conv2D( filters: 64,  kernel_size: (3, 3), activation='relu'))
model.add(layers.MaxPooling2D( pool_size: (2, 2), padding='same'))
model.add(layers.Conv2D( filters: 128,  kernel_size: (3, 3), activation='relu'))
model.add(layers.MaxPooling2D( pool_size: (2, 2), padding='same'))
model.add(layers.Flatten())
model.add(layers.Dense( units: 11, activation='softmax'))

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

Image 3. The model

5. Progress of results

5.1. Initial accuracy

The model initially achieved an accuracy of approximately 40%, providing a baseline for further improvements.

5.2. Train-test split

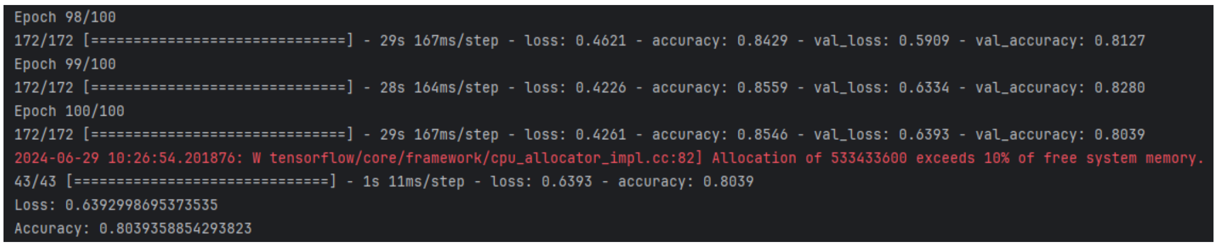
Adjusting the train-test split improved the model's performance to around 65%, indicating better generalization to unseen data.

5.3. Model architecture optimization

Refining the convolutional and max-pooling layers resulted in an accuracy of approximately 72%. These adjustments helped the model to better capture and distinguish between different weather conditions.

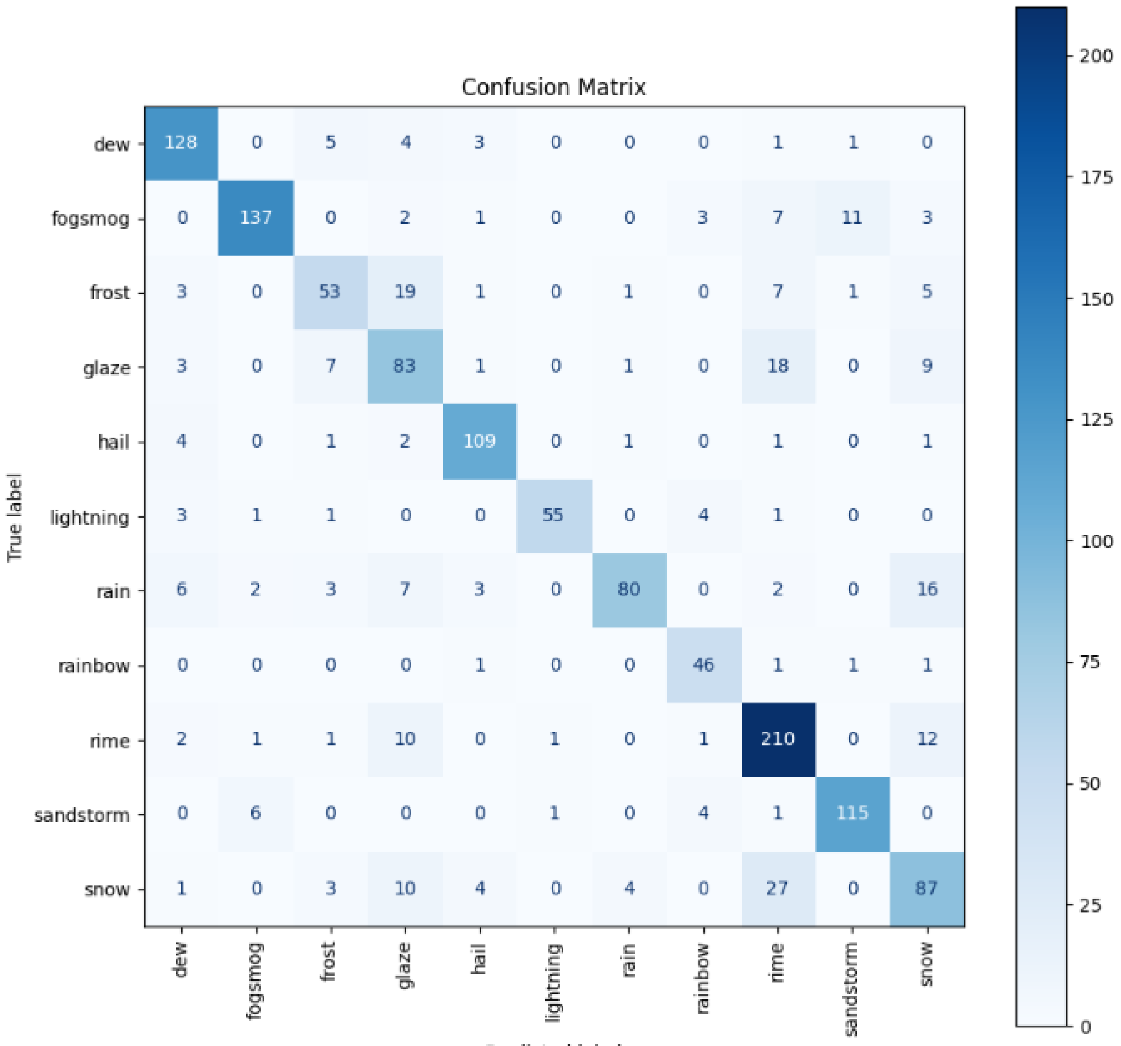
5.4. Data augmentation

Applying data augmentation techniques further enhanced the model's accuracy to around 80%. The augmented data provided a more diverse training set, enabling the model to learn more robust and generalized features.



```
Epoch 98/100
172/172 [==============================] - 29s 167ms/step - loss: 0.4621 - accuracy: 0.8429 - val_loss: 0.5909 - val_accuracy: 0.8127
Epoch 99/100
172/172 [==============================] - 28s 164ms/step - loss: 0.4226 - accuracy: 0.8559 - val_loss: 0.6334 - val_accuracy: 0.8280
Epoch 100/100
172/172 [==============================] - 29s 167ms/step - loss: 0.4261 - accuracy: 0.8546 - val_loss: 0.6393 - val_accuracy: 0.8039
2024-06-29 10:26:54.201876: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 533433600 exceeds 10% of free system memory.
43/43 [==============================] - 1s 11ms/step - loss: 0.6393 - accuracy: 0.8039
Loss: 0.6392998695373535
Accuracy: 0.8039358854293823
```

Image 4. Model’s accuracy after 100 epochs



	dew	fogsmog	frost	glaze	hail	lightning	rain	rainbow	rime	sandstorm	snow
dew	128	0	5	4	3	0	0	0	1	1	0
fogsmog	0	137	0	2	1	0	0	3	7	11	3
frost	3	0	53	19	1	0	1	0	7	1	5
glaze	3	0	7	83	1	0	1	0	18	0	9
hail	4	0	1	2	109	0	1	0	1	0	1
lightning	3	1	1	0	0	55	0	4	1	0	0
rain	6	2	3	7	3	0	80	0	2	0	16
rainbow	0	0	0	0	1	0	0	46	1	1	1
rime	2	1	1	10	0	1	0	1	210	0	12
sandstorm	0	6	0	0	0	1	0	4	1	115	0
snow	1	0	3	10	4	0	4	0	27	0	87

Image 5. Confusion matrix

6. Confusion matrix
   The most interesting things about the data presented on the confusion matrix is that snow, rime, rain and glaze were often confused with each other. It is most likely because there are similar colors on the images and even sometimes for people it is hard to recognize the weather correctly.

7. Trials
   Before achieving the final validation accuracy on the level of about 80% many different things were done. Firstly the dataset was manually divided into to folders – dataset and testset. The problem was it was not an 80-20 division but something around it. That is why the train_test_split was used. Then the problem that was noticed was overfitting. When there were more the 12 epochs, loss was decreasing but the validation loss was rising. It happened because the dataset was not big enough, so the data augmentation was used. There were also trials of adding dropout layers to prevent overfitting but there were not any significant changes. Another thing done to increase model's accuracy was to enlarge the number of epochs to about 150. Unfortunately after about 90 epochs the validation accuracy did not rise at all. Last thing but most time-consuming was adjusting the layers in

model. Different configurations were tried but none of them offered better performance than the one presented earlier.

8. Summary

   The machine learning is a powerful tool. It can make many things easier in people’s lives. However it can not be fully trusted. As we can see, in this project there is 20% that the model is wrong. We always have to take into consideration that the thing that model tells as might be wrong.

9. References

GitHub:
https://github.com/pawelledwon/Weather-Recognition/tree/master

https://www.kaggle.com/datasets/jehanbhathena/weather-dataset
https://pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/
https://www.tensorflow.org/install/pip