GSLC 2 – Deep Learning

Final Project Analysis Report

Nama : Fiona Maharani Nugraha

NIM : 2602199582

---

1. **Preprocessing methods, including data splitting, etc.**

- Data Splitting

```
[ ] #split data
train_data, val_data = train_test_split(data, test_size = 0.3, random_state = 10)

```

The dataset is split into training and validation sets with a ratio of 70:30. This is done using the `train_test_split` function from `sklearn.model_selection`.

- Data Augmentation and Rescaling

```
train_datagen = ImageDataGenerator(rescale = 1.0 / 255)

```

Image data is rescaled by 1/255 to normalize the pixel values to the range [0, 1].

- Data Loading

```
train_set = train_datagen.flow_from_dataframe(
    dataframe = train_data,
    x_col = "path",
    y_col = "label",
    target_size = (180, 180),
    batch_size = 16,
    class_mode = "sparse"
    )

val_datagen = ImageDataGenerator(rescale = 1.0 / 255)

val_set = train_datagen.flow_from_dataframe(
    dataframe = val_data,
    x_col = "path",
    y_col = "label",
    target_size = (180, 180),
    batch_size = 16,
    class_mode = "sparse"
)

```

Training and validation data are loaded using `flow_from_dataframe` which reads the images and corresponding labels from the dataframes. Images are resized to 180x180 pixels and loaded in 16 batches.

2. Model used

Our project designed and trained a Convolutional Neural Network (CNN) model using the Keras Sequential.

```python
model = Sequential() #layer linear
model.add(Input(shape = (180, 180, 3))) #menambahkan layer input
#mulai CNN -> feature map
model.add(layers.Conv2D(64, (3, 3), activation = "relu"))
#menambahkan layer convolution
#ada 64 filter, isinya matrix 3 x 3
model.add(layers.MaxPooling2D(2,2))
#32 filter dengan matriks 3x3
model.add(layers.Conv2D(32, (3,3), activation = "relu"))
model.add(layers.MaxPooling2D(2, 2)) #ngurangin dimensi spasial output dari layer sebelumnya
#akhir CNN

model.add(layers.Flatten()) #ngubah tensor dari 2D -> 1D
model.add(layers.Dense(512, activation = "relu"))
model.add(layers.Dense(256, activation = "relu")) #menambahkan layer dense
model.add(layers.Dense(bt_class_num, activation = "softmax")) #

model.compile(
    loss = 'sparse_categorical_crossentropy', #
    optimizer = tf.keras.optimizers.RMSprop(learning_rate = 0.0005), #optimalkan model
    metrics = ["accuracy"]
)

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 178, 178, 64)      1792

 max_pooling2d_1 (MaxPoolin  (None, 89, 89, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 87, 87, 32)        18464

 max_pooling2d_2 (MaxPoolin  (None, 43, 43, 32)        0
 g2D)

 flatten_1 (Flatten)         (None, 59168)             0

 dense_2 (Dense)             (None, 512)               30294528

 dense_3 (Dense)             (None, 256)               131328

 dense_4 (Dense)             (None, 75)                19275

=================================================================
Total params: 30465387 (116.22 MB)
Trainable params: 30465387 (116.22 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

This CNN model leverages multiple convolutional and dense layers to classify images into one of the predefined classes. By using pooling layers, the model reduces computational complexity and the risk of overfitting. The final dense layers with ReLU activations allow the network to learn complex patterns, while the softmax output layer ensures the model produces probabilities for each class, facilitating multi-class classification.

3. Training and validation results

```python
model_1 = model.fit(
    train_set,
    epochs = 10,
    validation_data = val_set
)

#4945/16 = 285
#setiap batch 16 gambar
```

```
Epoch 1/10
285/285 [==============================] - 1325s 5s/step - loss: 5.6096 - accuracy: 0.0736 - val_loss: 3.2236 - val_accuracy: 0.2387
Epoch 2/10
285/285 [==============================] - 950s 3s/step - loss: 2.6802 - accuracy: 0.3464 - val_loss: 1.4194 - val_accuracy: 0.6711
Epoch 3/10
285/285 [==============================] - 1003s 4s/step - loss: 1.3138 - accuracy: 0.6685 - val_loss: 0.4607 - val_accuracy: 0.9209
Epoch 4/10
285/285 [==============================] - 967s 3s/step - loss: 0.4426 - accuracy: 0.9022 - val_loss: 0.1598 - val_accuracy: 0.9760
Epoch 5/10
285/285 [==============================] - 996s 3s/step - loss: 0.1143 - accuracy: 0.9804 - val_loss: 0.0223 - val_accuracy: 0.9991
Epoch 6/10
285/285 [==============================] - 976s 3s/step - loss: 0.0206 - accuracy: 0.9978 - val_loss: 0.0213 - val_accuracy: 0.9965
Epoch 7/10
285/285 [==============================] - 997s 3s/step - loss: 0.0085 - accuracy: 0.9985 - val_loss: 0.0029 - val_accuracy: 0.9998
Epoch 8/10
285/285 [==============================] - 994s 3s/step - loss: 0.0017 - accuracy: 0.9998 - val_loss: 4.5579e-04 - val_accuracy: 1.0000
Epoch 9/10
285/285 [==============================] - 1009s 4s/step - loss: 4.6892e-04 - accuracy: 1.0000 - val_loss: 1.8578e-04 - val_accuracy: 1.0000
Epoch 10/10
285/285 [==============================] - 955s 3s/step - loss: 1.7547e-04 - accuracy: 1.0000 - val_loss: 1.0367e-04 - val_accuracy: 1.0000
```

Model was trained for 10 epochs, showing significant improvement in both training and validation accuracy. Starting with a low accuracy of 7.36% in the first epoch, the model achieved perfect accuracy by the tenth epoch. Training and validation losses decreased dramatically, indicating effective learning and convergence. Despite the model's excellent performance on both training and validation sets, the near-perfect results suggest a potential risk of overfitting.

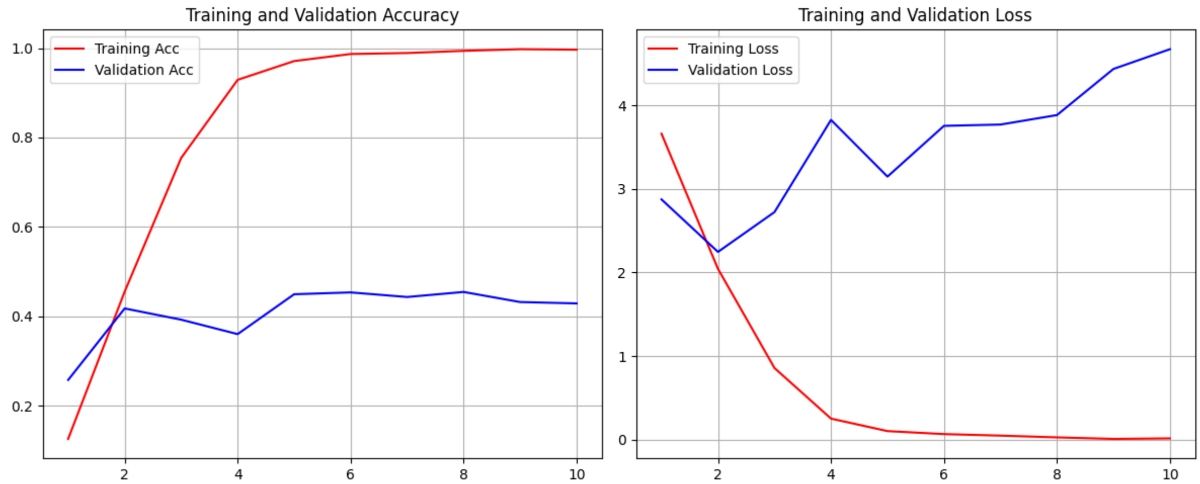
4. Evaluation results

```python
test_loss, test_accuracy = model.evaluate(test_set)
print(f"Test Accuracy: {test_accuracy}")
print(f"Test Loss: {test_loss}")
```

```
285/285 [==============================] - 100s 349ms/step - loss: 0.0032 - accuracy: 0.9989
Test Accuracy: 0.998900830745697
Test Loss: 0.0031990171410143375
```

Both test loss and test accuracy suggest that the model is performed well on the test set. Low loss means that the model's predictions are close the true labels, while the high accuracy shows that almost all predictions are correct.

However, the plot shows that the model is overfitted.



The training accuracy increases significantly while the validation accuracy does not increases significantly. Meanwhile the loss plot has decreasing training loss but increasing validation loss. Model is overfitted because the divergence between training and validation metrics. The model is learning the specific patterns of the training data rather than the underlying distribution that applies to both the training and validation datasets.

5. Existing challenges or future opportunities for model

Class Imbalances	The count plot indicates that some butterfly classes are underrepresented. Techniques like data augmentation, oversampling, or class weighting could be used to address this imbalance.
Model complexity	Depending on the performance, the model complexity can be increased by adding more layers or using pre-trained models like VGG16..
Data Augmentation	Implementing more robust data augmentation techniques can help in improving the model's generalization capabilities.
Hyperparameter Tuning	Experiment with different hyperparameters such as learning rate, batch size, and optimizer to find the optimal configuration.
Early Stopping	Since the model was overfit, implementing this callback might help to avoid overfitting since it

	monitors the validation loss and stop training when it stops improving to the training data.

Difference between Image Segmentation vs Object Detection:

- Image segmentation model focus on dividing or segmenting where each pixel in a image gets its own color based on what it represents.
- Meanwhile object detection models are like finding and marking where specific things are in a picture using boxes and labels.