

Practical 8: Overfitting – Dropout Technique

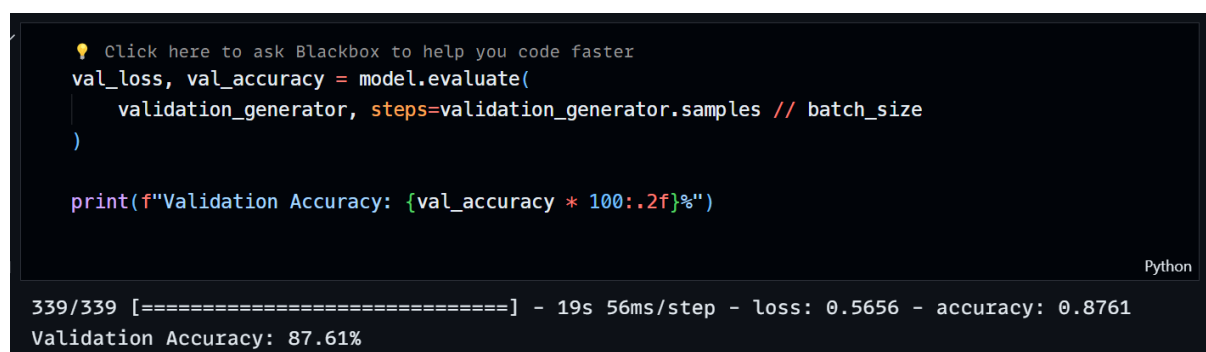
Overfitting and Regularization in Plant Disease Detection CNN

One of the key challenges in training deep learning models, particularly Convolutional Neural Networks (CNNs), is overfitting. Overfitting occurs when a model becomes too specialized on the training data and fails to generalize well to unseen data. This manifests as high training accuracy but low validation accuracy. In the context of plant disease detection, an overfitting model might achieve impressive results on the training images but struggle to accurately diagnose diseases in new images with slightly different characteristics.

To mitigate overfitting and improve the generalizability of our Plant Disease detection CNN model, we can employ a regularization technique called dropout. Dropout randomly drops out a certain percentage of neurons during training. This forces the model to learn redundant representations and prevents it from relying on any single neuron or its specific quirks in the training data. By encountering different "reduced versions" of the training data during each epoch, the model is encouraged to learn more generalizable features that are robust to variations in the data.

Before Dropout Implementation

```
model.add(layers.MaxPooling2D(2, 2))
model.add(layers.Conv2D(64, (3, 3), activation="relu"))
model.add(layers.MaxPooling2D(2, 2))
model.add(layers.Flatten())
model.add(layers.Dense(256, activation="relu"))
model.add(layers.Dense(train_generator.num_classes,
activation="softmax"))
```



```
Click here to ask Blackbox to help you code faster
val_loss, val_accuracy = model.evaluate(
    validation_generator, steps=validation_generator.samples // batch_size
)


print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
```

Python

339/339 [=====] - 19s 56ms/step - loss: 0.5656 - accuracy: 0.8761
Validation Accuracy: 87.61%

After Dropout Implementation

```
model.add(layers.Conv2D(32, (3, 3), activation="relu",  
input_shape=(img_size, img_size, 3)))  
  
model.add(layers.MaxPooling2D(2, 2))  
  
model.add(layers.Dropout(0.25))  
  
model.add(layers.Conv2D(64, (3, 3), activation="relu"))  
  
model.add(layers.MaxPooling2D(2, 2))  
  
model.add(layers.Dropout(0.25))
```



```
Click here to ask Blackbox to help you code faster  
val_loss, val_accuracy = model.evaluate(  
    validation_generator, steps=validation_generator.samples // batch_size  
)  
  
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
```

Python

339/339 [=====] - 18s 53ms/step - loss: 0.3131 - accuracy: 0.9100
Validation Accuracy: 91.00%

In this example, Dropout(0.25) is used after each convolutional layer. This means that during training, 25% of the neurons in those layers will be randomly dropped out. The dropout rate is a hyperparameter that can be tuned for optimal performance. Experimenting with different dropout rates (between 0.2 and 0.5) can help achieve the best balance between preventing overfitting and maintaining model accuracy.

The dropout rate (e.g., 0.25 in the code) determines the percentage of neurons dropped in each layer during training. It's a hyperparameter you can tune. Here's a general guideline:

- **Low Dropout (0-0.2):** Might not be effective enough in preventing overfitting.
- **Moderate Dropout (0.2-0.5):** Often a good starting point for many CNN architectures.
- **High Dropout (0.5-0.8):** Can be too aggressive and might hurt performance if not tuned carefully.

By incorporating dropout as a regularization technique, we can enhance the robustness and generalizability of our Plant Disease detection CNN model. This allows the model to perform more effectively on unseen plant images, ultimately leading to more accurate disease diagnoses.