Fine-tuning

Ramin Zarebidoky (LiterallyTheOne)

20 Dec 2025

Fine-tuning

Introduction

In the previous tutorials, we learned about **transfer learning** and how to use the advantage of a pre-trained model on our dataset. In this Tutorial, we learn about how to apply fine-tuning and see the results. Also, we explain the concepts of **Underfitting** and **Overfitting** and how to solve them. Finally, we make more classification layer more generalized.

Fine-tuning

Fine-tuning is a technique in **Deep Learning** that we use to adapt our pre-trained model with the new **Dataset**. In the previous tutorials, we worked with **transfer learning**. **Fine-tuning** is pretty similar to **transfer learning**. The only difference is that we unfreeze some of the last layers to our **base_model** in order to train them. Here is an example:

```
base_model = MobileNetV2(include_top=False, input_shape=(224,
 ↪  224, 3))

for layer in base_model.layers[:-4]:
    layer.trainable = False
```

In the code above, we froze the starting layers of our `base_model` and left the last 4 layers as `trainable`. Now, let's print the `base_model` summary with `show_trainable=True` like below:

```
print(base_model.summary(show_trainable=True))

"""
--------
output:

  ...
```

```
| block_16_project    (None, 7, 7,        307,200   block_16_dept…
↪      N
 (Conv2D)            320)
↪

| block_16_project…   (None, 7, 7,          1,280   block_16_proj…
↪      Y
 (BatchNormalizat…   320)
↪

 Conv_1 (Conv2D)     (None, 7, 7,        409,600   block_16_proj…
↪  Y
                     1280)
↪

 Conv_1_bn           (None, 7, 7,          5,120   Conv_1[0][0]
↪  Y
 (BatchNormalizat…   1280)
↪

 out_relu (ReLU)     (None, 7, 7,              0   Conv_1_bn[0][…
↪  -
                     1280)
↪


Total params: 2,257,984 (8.61 MB)
Trainable params: 412,800 (1.57 MB)
Non-trainable params: 1,845,184 (7.04 MB)

"""
```

As you can see, the last 4 layers, are trainable. The only thing that we should do, is to train our model like before.

Underfitting

Underfitting happens when our model is not training well on our **training data**. In other words, our model is not capable of learning the pattern of our data. There are different reasons that might cause this phenomenon to happen. One of the most important ones is that our model is too simple for the problem that we have. To solve this problem, we should choose a more complex model with more trainable layers.

Another reason behind **Underfitting**, is that we used too much **Regulariza-**

tion. For example, we have used so many **Augmentation** layers. To solve it, we should just choose the suited **Regularization** techniques.

Sometimes, we haven't chosen the correct input features required for understanding the pattern. For example, if we want to estimate the house price, and we don't have the size of the house, our model is not going to figure out the pattern.

Overfitting

Overfitting happens when our model is doing well on training data but the results on unseen data (**Validation** and **Test**) are not good. In other words, model has understood the pattern so well (including the noise), but it fails to generalize. There are some reasons that might be the cause of **Overfitting**. One of the most important ones is that our model is too complex for the dataset that we have. To fix this problem, we should choose a simpler model or lower the number of trainable layers.

Another reason is that, we train our model for a long time. To solve this problem, we said that we can use **EarlyStopping**.

One of the other reasons is that, we don't use enough regularization. For example, if our data is the images taken on the nature with so many different contrast and brightness, it is expected that our unseen data is also has this differences. So, if we want our model to learn how to deal with them, we should use the respective **Augmentation**.

Make our classification layer more generalized

In the previous tutorials, we only used a **Fully Connected** layer for our **Classification layer**. Now, let's change the **Classification layer** based on the things that we learned. Here is an example:

```python
model = keras.Sequential(
    [
        ...,
        layers.GlobalAveragePooling2D(),
        layers.Dropout(0.5),
        layers.Dense(128, activation="relu"),
        layers.Dropout(0.5),
        layers.Dense(4, activation="softmax"),
    ]
)
```

As you can see, in the code above, at first, I used a `GlobalAveragePooling2D` instead of `Flatten`. It helps the model to be more generalized. Then I used a `dropout` layer. This layer helps to have more generalization. Instead of using only one layer, I have used 2 layers. The first layer has 128 neurons with the activation of `relu` and the other layer has 4 neurons with the activation of `softmax` in order to guess the classification.

Your turn

Now, Change the **transfer learning** to **fine-tuning** and make your classification layer more generalized.

Conclusion

In this Tutorial, at first, we explained how we can apply **fine-tuning**. Then, we described **Underfitting** and **Overfitting** and how to solve them. Finally, we made our classification layer more generalized.