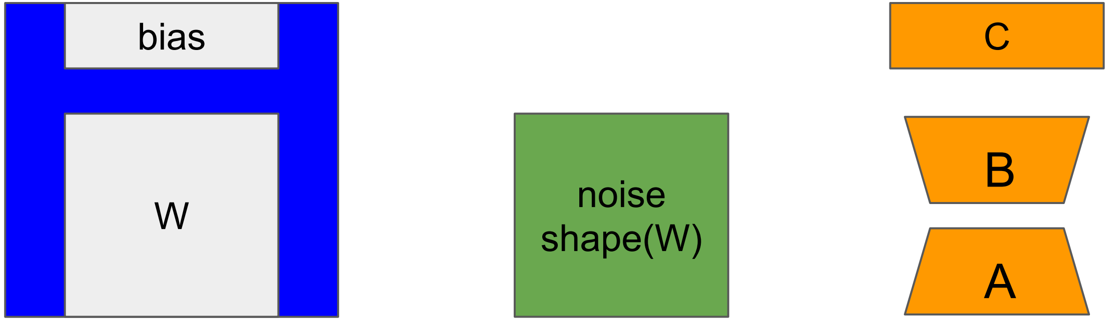
5주차 (11/15~11/22)

3L

Scheduling, Noise, Bias 그림은 중요하다

decay factor



bias

W

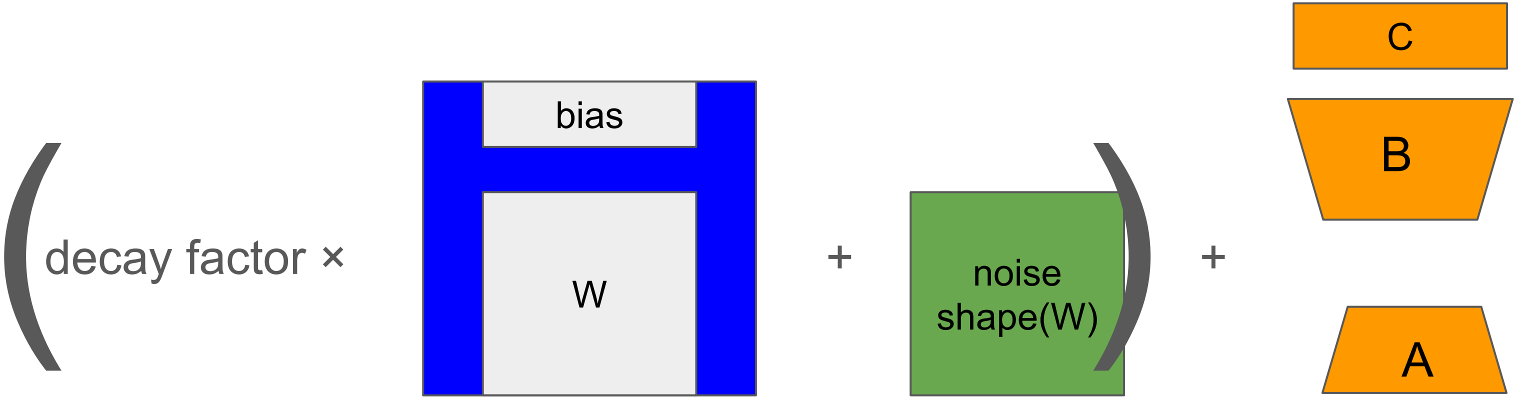
noise shape(W)

C

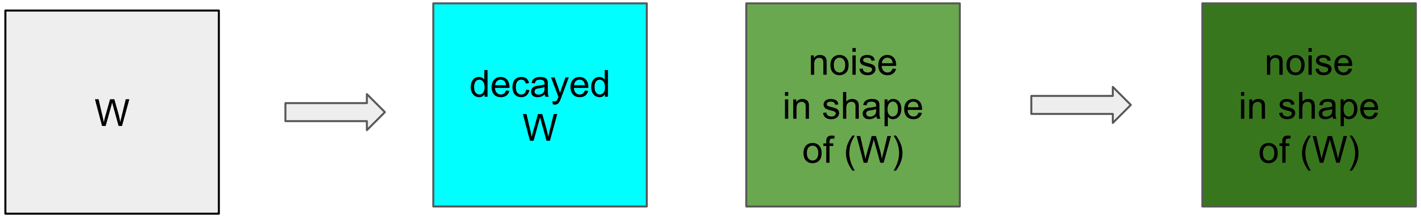
B

A

Scheduling, Noise, Bias 그림은 중요하다



파라미터의 평균은 m
분산을 V



W

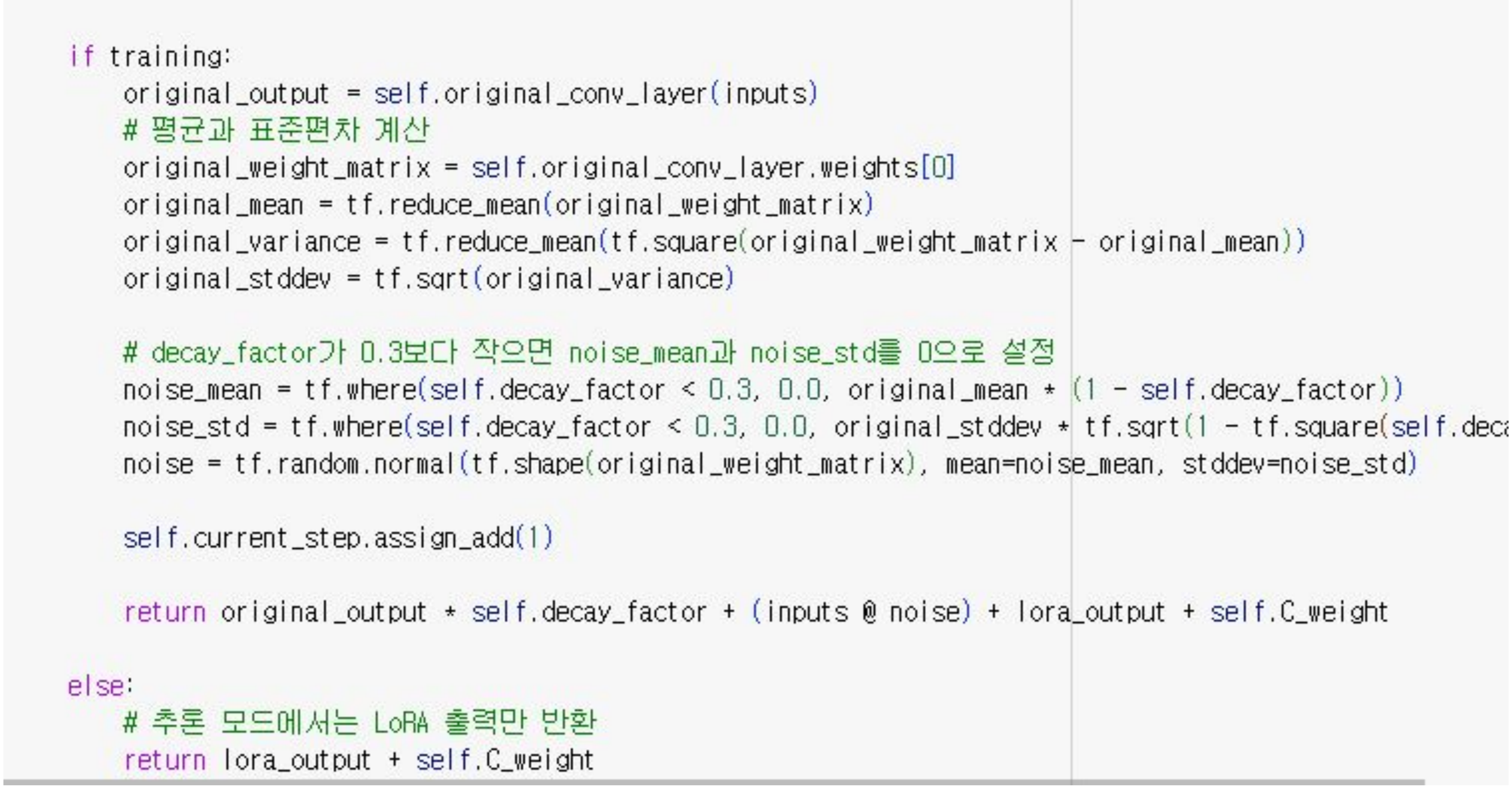
decayed
W

noise
in shape
of (W)

noise
in shape
of (W)

W 의 파라미터의 평균은 m
분산을 V

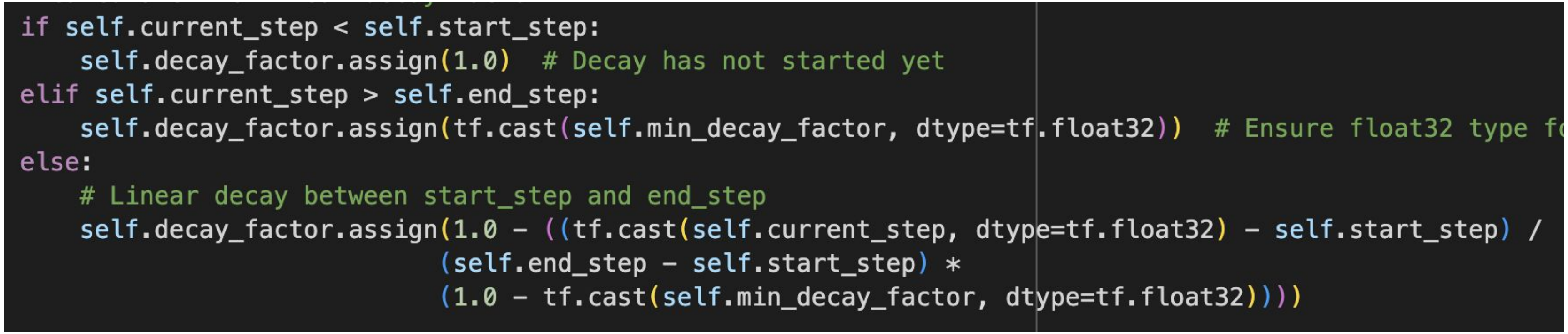
파라미터의 평균은 m
분산을 V



```
if training:
    original_output = self.original_conv_layer(inputs)
    # 평균과 표준편차 계산
    original_weight_matrix = self.original_conv_layer.weights[0]
    original_mean = tf.reduce_mean(original_weight_matrix)
    original_variance = tf.reduce_mean(tf.square(original_weight_matrix - original_mean))
    original_stddev = tf.sqrt(original_variance)

    # decay_factor가 0.3보다 작으면 noise_mean과 noise_std를 0으로 설정
    noise_mean = tf.where(self.decay_factor < 0.3, 0.0, original_mean * (1 - self.decay_factor))
    noise_std = tf.where(self.decay_factor < 0.3, 0.0, original_stddev * tf.sqrt(1 - tf.square(self.deca
    noise = tf.random.normal(tf.shape(original_weight_matrix), mean=noise_mean, stddev=noise_std)

    self.current_step.assign_add(1)

    return original_output * self.decay_factor + (inputs @ noise) + lora_output + self.C_weight

else:
    # 추론 모드에서는 LoRA 출력만 반환
    return lora_output + self.C_weight
```

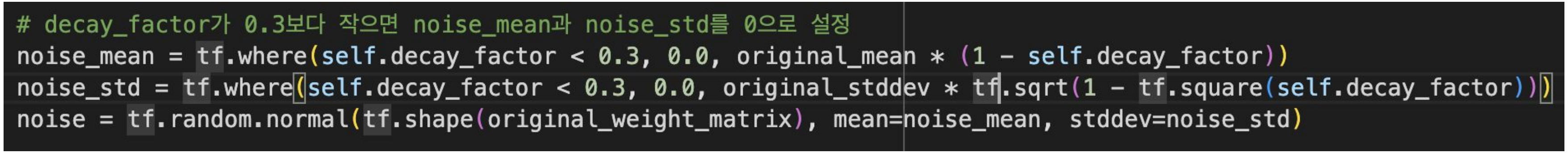
Scheduling

decay factor schedule: warm up + linear decay + cool down



```
if self.current_step < self.start_step:
    self.decay_factor.assign(1.0)  # Decay has not started yet
elif self.current_step > self.end_step:
    self.decay_factor.assign(tf.cast(self.min_decay_factor, dtype=tf.float32))  # Ensure float32 type fo
else:
    # Linear decay between start_step and end_step
    self.decay_factor.assign(1.0 - ((tf.cast(self.current_step, dtype=tf.float32) - self.start_step) /
                            (self.end_step - self.start_step) *
                            (1.0 - tf.cast(self.min_decay_factor, dtype=tf.float32))))
```

noise schedule: compensate stage + clear stage



```
# decay_factor가 0.3보다 작으면 noise_mean과 noise_std를 0으로 설정
noise_mean = tf.where(self.decay_factor < 0.3, 0.0, original_mean * (1 - self.decay_factor))
noise_std = tf.where(self.decay_factor < 0.3, 0.0, original_stddev * tf.sqrt(1 - tf.square(self.decay_factor)))
noise = tf.random.normal(tf.shape(original_weight_matrix), mean=noise_mean, stddev=noise_std)
```

Noise

decay factor로 감소 해도 전체의 평균과 표준편차를 original layer 의 평균과 표준편차로 유지하는 Noise(normal distribution)를 선정.

original layer * decay factor

가정: noise 와 original weight는 서로 독립적이다

전체 분산 = 노이즈 분산 + 감소된 분산 (가정에 의해 성립)

m=mean(W)
δ=std(W)
d=decay factor

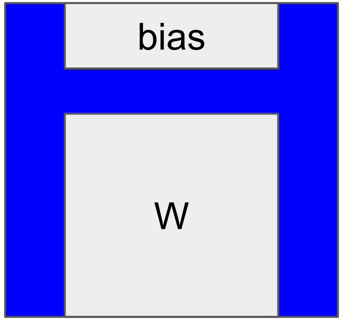
$\text{noise} = N(\ (1 - d)m,\ (1-d^2)\delta^2\ )$

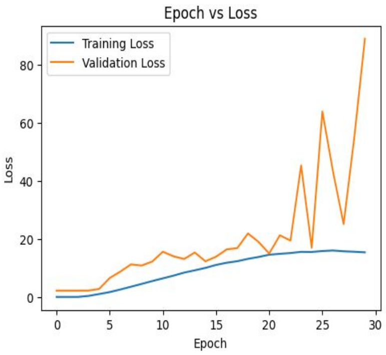
Bias

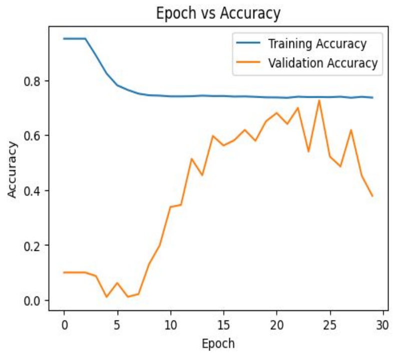
decay of original output -> decay of weight & bias

need to compensate with new bias C

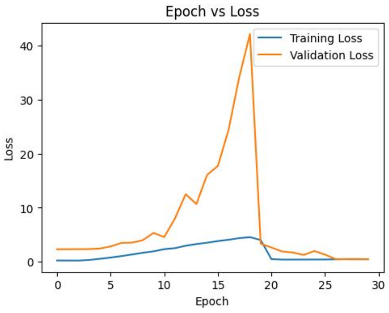
decay factor ×

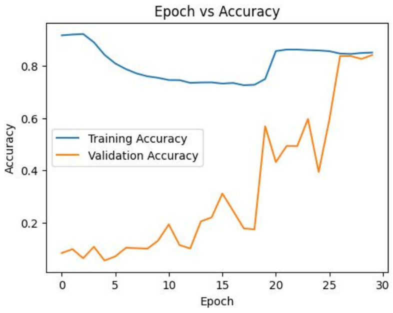




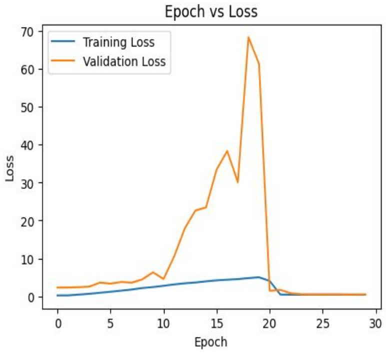


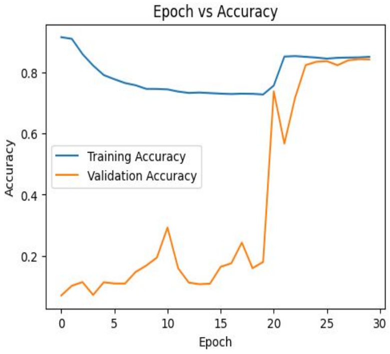
decay factor schedule: start(0.1), end(0.9)
Noise schedule : increase until 0.9 & maintain
weight C: none



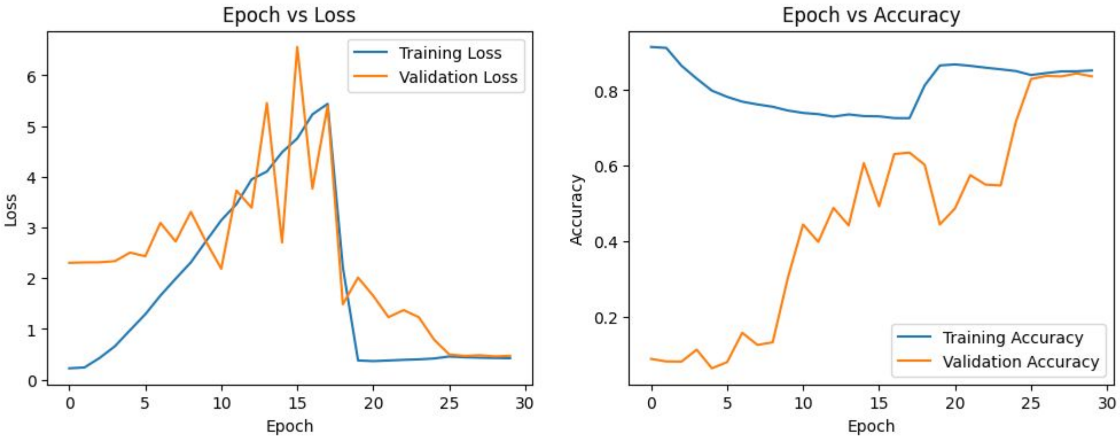


decay factor schedule: start(0.1), end(0.9)
Noise schedule : increase before 0.3d & vanish
weight C (0)

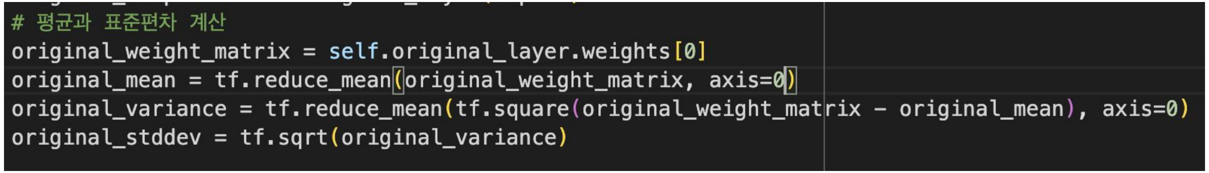




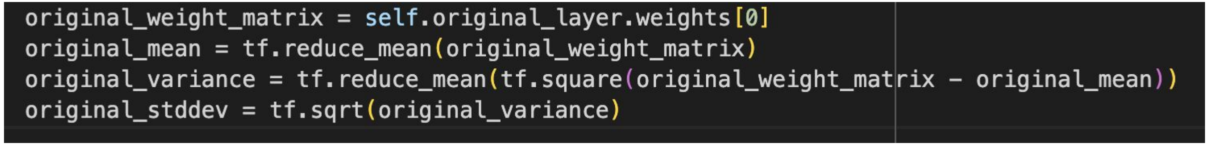
decay factor schedule: start(0.05), end(0.85)
Noise schedule : increase before 0.2d & vanish
weight C (0)


Epoch vs Loss

Epoch vs Accuracy



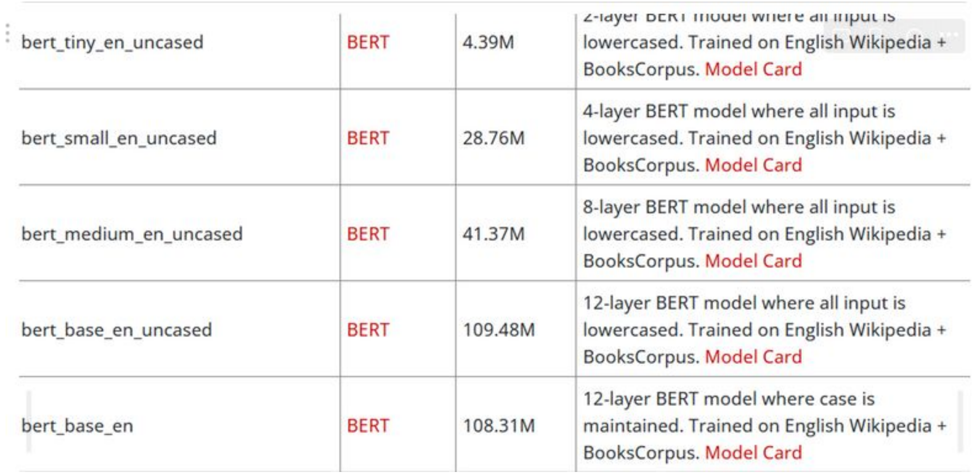
```
# 평균과 표준편차 계산
original_weight_matrix = self.original_layer.weights[0]
original_mean = tf.reduce_mean(original_weight_matrix, axis=0)
original_variance = tf.reduce_mean(tf.square(original_weight_matrix - original_mean), axis=0)
original_stddev = tf.sqrt(original_variance)
```



```
original_weight_matrix = self.original_layer.weights[0]
original_mean = tf.reduce_mean(original_weight_matrix)
original_variance = tf.reduce_mean(tf.square(original_weight_matrix - original_mean))
original_stddev = tf.sqrt(original_variance)
```

Implementation in a larger model

- Amazon Review Polarity Dataset (IMDB 로도 해봤었음)
- Binary Text Classification Task with Bert
-

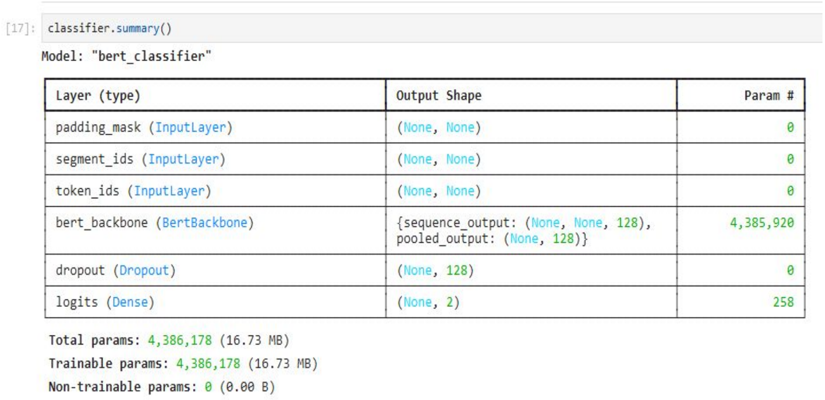


bert_tiny_en_uncased	BERT	4.39M	2-layer BERT model where all input is lowercased. Trained on English Wikipedia + BooksCorpus. Model Card
bert_small_en_uncased	BERT	28.76M	4-layer BERT model where all input is lowercased. Trained on English Wikipedia + BooksCorpus. Model Card
bert_medium_en_uncased	BERT	41.37M	8-layer BERT model where all input is lowercased. Trained on English Wikipedia + BooksCorpus. Model Card
bert_base_en_uncased	BERT	109.48M	12-layer BERT model where all input is lowercased. Trained on English Wikipedia + BooksCorpus. Model Card
bert_base_en	BERT	108.31M	12-layer BERT model where case is maintained. Trained on English Wikipedia + BooksCorpus. Model Card

- 이해할 수 없는 현상 :
- (IMDB Dataset 에서)
  Bert Tiny 에서 98% (Val acc) 까지 올라가던 성능이, Bert Small 로 모델 크기를 키웠더니 학습이 되지 않는 현상 발생함. , 50% (Val acc) 로 계속 Fix 된 값이 나왔었음.
- 그래서 Dataset 을 변경하게 됨.

Replace Bert Classifier with LoRA

Bert Tiny

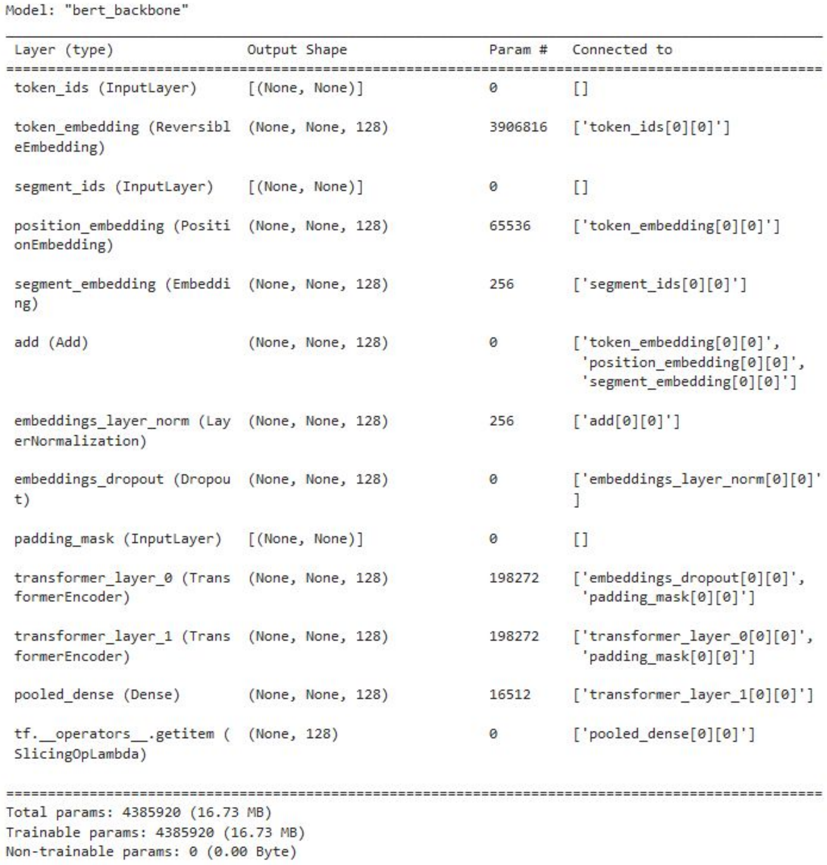


```
[17]: classifier.summary()
```

Model: "bert_classifier"

Layer (type)	Output Shape	Param #
padding_mask (InputLayer)	(None, None)	0
segment_ids (InputLayer)	(None, None)	0
token_ids (InputLayer)	(None, None)	0
bert_backbone (BertBackbone)	{sequence_output: (None, None, 128), pooled_output: (None, 128)}	4,385,920
dropout (Dropout)	(None, 128)	0
logits (Dense)	(None, 2)	258

Total params: 4,386,178 (16.73 MB)
Trainable params: 4,386,178 (16.73 MB)
Non-trainable params: 0 (0.00 B)



```
Model: "bert_backbone"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 token_ids (InputLayer)         [(None, None)]       0           []

 token_embedding (Reversibl     (None, None, 128)    3906816     ['token_ids[0][0]']
 eEmbedding)

 segment_ids (InputLayer)       [(None, None)]       0           []

 position_embedding (Positi     (None, None, 128)    65536       ['token_embedding[0][0]']
 onEmbedding)

 segment_embedding (Embeddi     (None, None, 128)    256         ['segment_ids[0][0]']
 ng)

 add (Add)                      (None, None, 128)    0           ['token_embedding[0][0]',
                                                                  'position_embedding[0][0]',
                                                                  'segment_embedding[0][0]']

 embeddings_layer_norm (Lay     (None, None, 128)    256         ['add[0][0]']
 erNormalization)

 embeddings_dropout (Dropou     (None, None, 128)    0           ['embeddings_layer_norm[0][0]'
 t)                                                              ]

 padding_mask (InputLayer)      [(None, None)]       0           []

 transformer_layer_0 (Trans     (None, None, 128)    198272      ['embeddings_dropout[0][0]',
 formerEncoder)                                                   'padding_mask[0][0]']

 transformer_layer_1 (Trans     (None, None, 128)    198272      ['transformer_layer_0[0][0]',
 formerEncoder)                                                   'padding_mask[0][0]']

 pooled_dense (Dense)           (None, None, 128)    16512       ['transformer_layer_1[0][0]']

 tf.__operators__.getitem (     (None, 128)          0           ['pooled_dense[0][0]']
 SlicingOpLambda)

==================================================================================================
Total params: 4385920 (16.73 MB)
Trainable params: 4385920 (16.73 MB)
Non-trainable params: 0 (0.00 Byte)
```

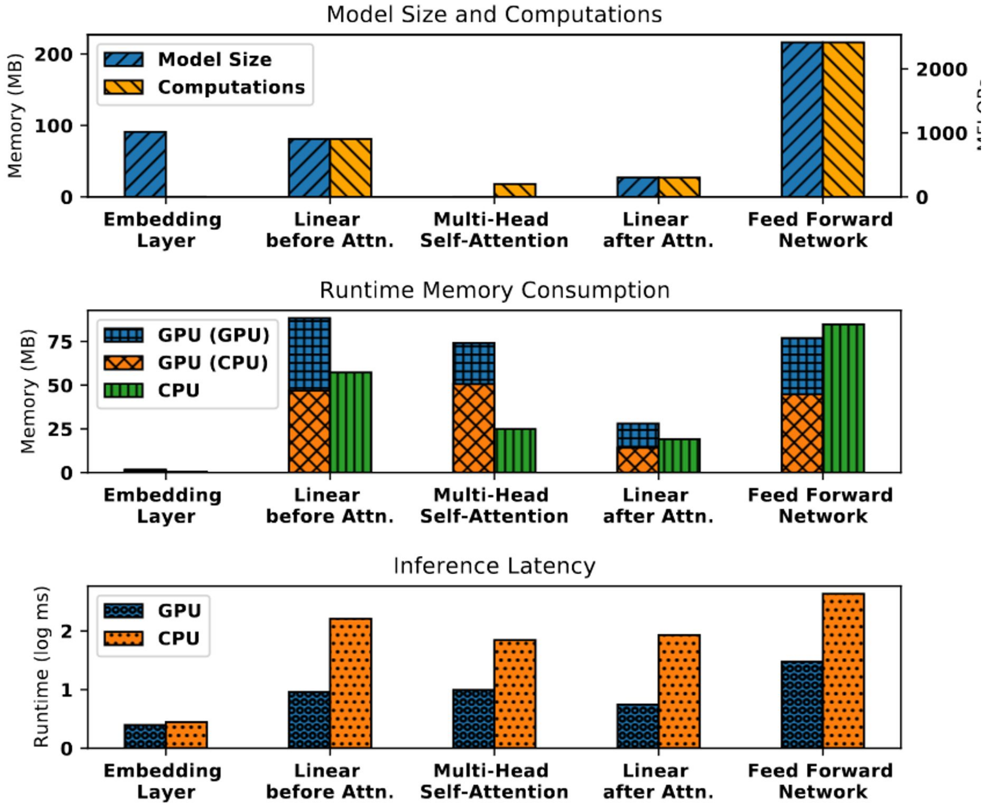
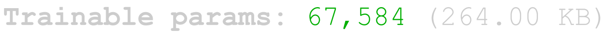


Figure 3: Breakdown Analysis of $\text{BERT}_{\text{BASE}}$.

rank 32

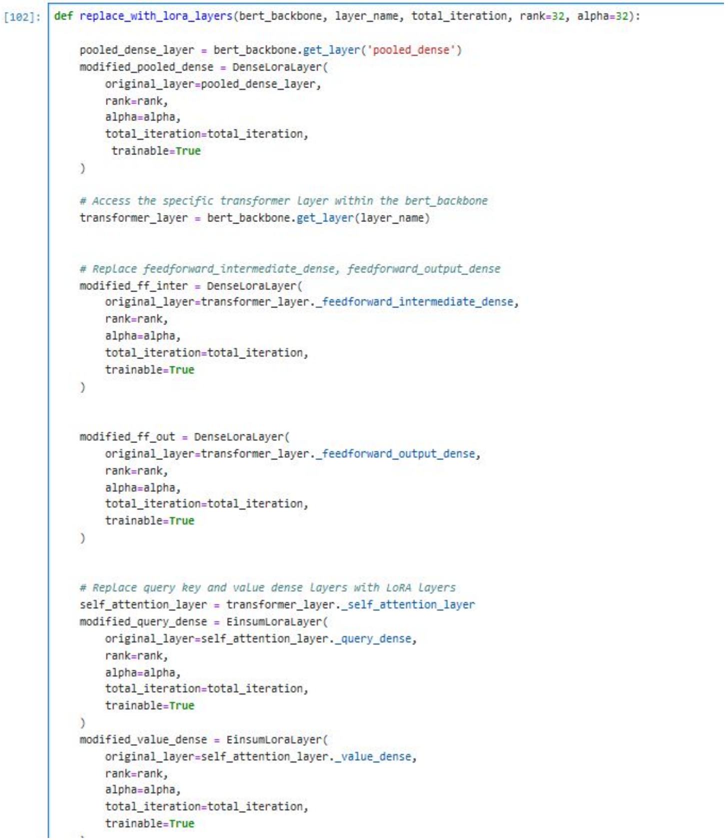


```
Trainable params: 67,584 (264.00 KB)
```

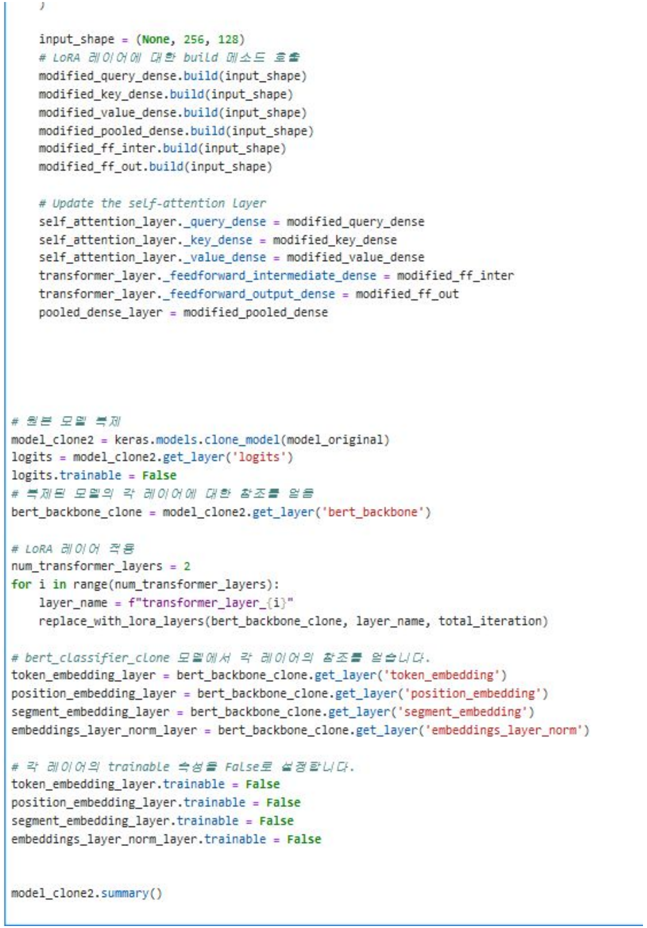
Bert Back Bone :

Before LoRA , After LoRA

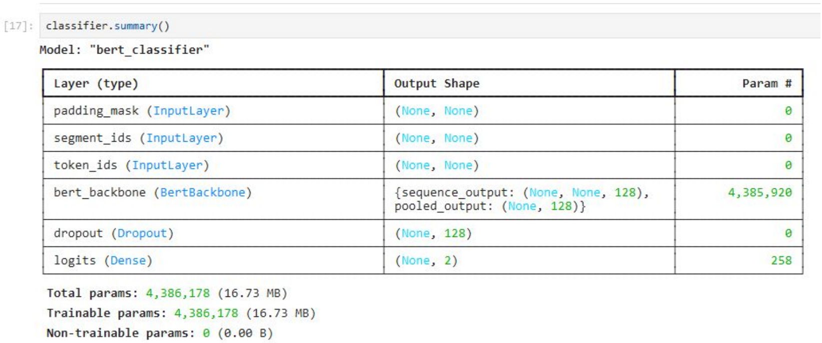
Replace Bert Classifier with LoRA



```
[102]: def replace_with_lora_layers(bert_backbone, layer_name, total_iteration, rank=32, alpha=32):

           pooled_dense_layer = bert_backbone.get_layer('pooled_dense')
           modified_pooled_dense = DenseLoraLayer(
               original_layer=pooled_dense_layer,
               rank=rank,
               alpha=alpha,
               total_iteration=total_iteration,
                trainable=True
           )

           # Access the specific transformer layer within the bert_backbone
           transformer_layer = bert_backbone.get_layer(layer_name)


           # Replace feedforward_intermediate_dense, feedforward_output_dense
           modified_ff_inter = DenseLoraLayer(
               original_layer=transformer_layer._feedforward_intermediate_dense,
               rank=rank,
               alpha=alpha,
               total_iteration=total_iteration,
               trainable=True
           )


           modified_ff_out = DenseLoraLayer(
               original_layer=transformer_layer._feedforward_output_dense,
               rank=rank,
               alpha=alpha,
               total_iteration=total_iteration,
               trainable=True
           )


           # Replace query key and value dense layers with LoRA layers
           self_attention_layer = transformer_layer._self_attention_layer
           modified_query_dense = EinsumLoraLayer(
               original_layer=self_attention_layer._query_dense,
               rank=rank,
               alpha=alpha,
               total_iteration=total_iteration,
               trainable=True
           )
           modified_value_dense = EinsumLoraLayer(
               original_layer=self_attention_layer._value_dense,
               rank=rank,
               alpha=alpha,
               total_iteration=total_iteration,
               trainable=True
```

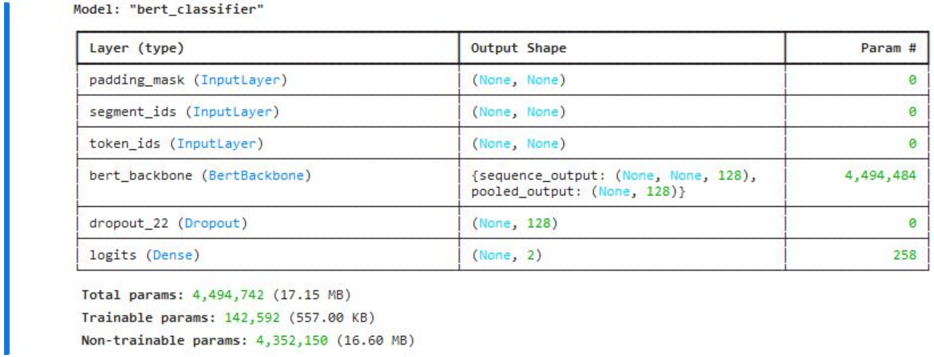


```
    )

    input_shape = (None, 256, 128)
    # LoRA 레이어에 대한 build 메소드 호출
    modified_query_dense.build(input_shape)
    modified_key_dense.build(input_shape)
    modified_value_dense.build(input_shape)
    modified_pooled_dense.build(input_shape)
    modified_ff_inter.build(input_shape)
    modified_ff_out.build(input_shape)

    # Update the self-attention layer
    self_attention_layer._query_dense = modified_query_dense
    self_attention_layer._key_dense = modified_key_dense
    self_attention_layer._value_dense = modified_value_dense
    transformer_layer._feedforward_intermediate_dense = modified_ff_inter
    transformer_layer._feedforward_output_dense = modified_ff_out
    pooled_dense_layer = modified_pooled_dense




# 원본 모델 복제
model_clone2 = keras.models.clone_model(model_original)
logits = model_clone2.get_layer('logits')
logits.trainable = False
# 복제된 모델의 각 레이어에 대한 참조를 얻음
bert_backbone_clone = model_clone2.get_layer('bert_backbone')

# LoRA 레이어 적용
num_transformer_layers = 2
for i in range(num_transformer_layers):
    layer_name = f"transformer_layer_{i}"
    replace_with_lora_layers(bert_backbone_clone, layer_name, total_iteration)

# bert_classifier_clone 모델에서 각 레이어의 참조를 얻습니다.
token_embedding_layer = bert_backbone_clone.get_layer('token_embedding')
position_embedding_layer = bert_backbone_clone.get_layer('position_embedding')
segment_embedding_layer = bert_backbone_clone.get_layer('segment_embedding')
embeddings_layer_norm_layer = bert_backbone_clone.get_layer('embeddings_layer_norm')

# 각 레이어의 trainable 속성을 False로 설정합니다.
token_embedding_layer.trainable = False
position_embedding_layer.trainable = False
segment_embedding_layer.trainable = False
embeddings_layer_norm_layer.trainable = False


model_clone2.summary()
```

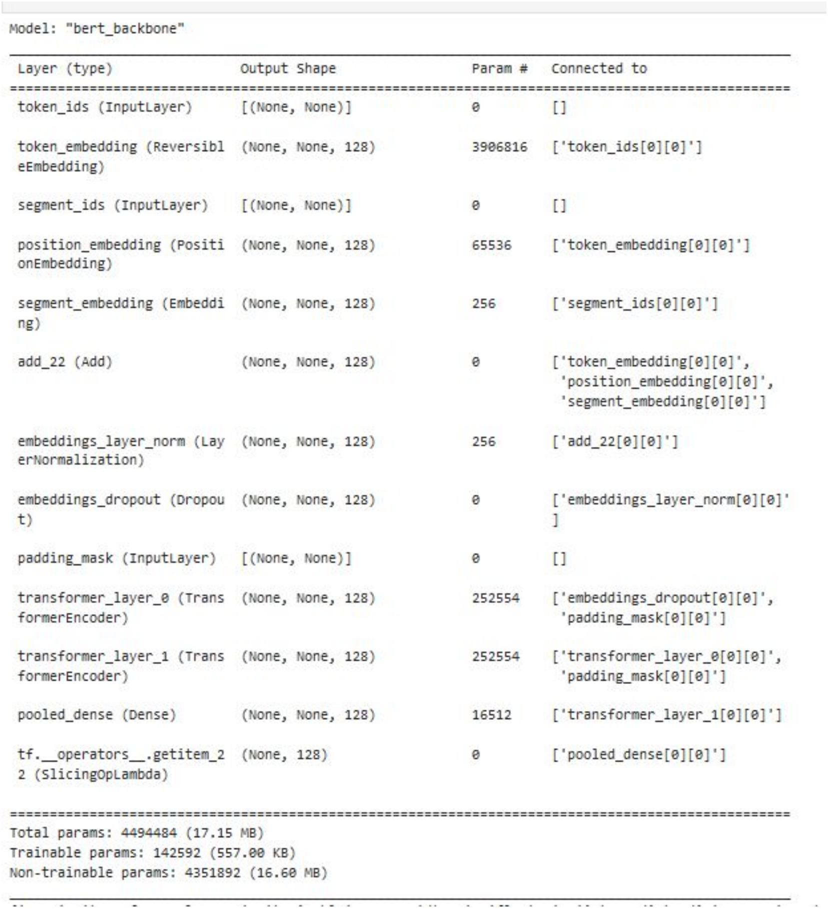


```
[17]: classifier.summary()
```

```
Model: "bert_classifier"
┌──────────────────────────────┬────────────────────────────────────────┬────────────┐
│ Layer (type)                 │ Output Shape                           │    Param # │
├──────────────────────────────┼────────────────────────────────────────┼────────────┤
│ padding_mask (InputLayer)    │ (None, None)                           │          0 │
├──────────────────────────────┼────────────────────────────────────────┼────────────┤
│ segment_ids (InputLayer)     │ (None, None)                           │          0 │
├──────────────────────────────┼────────────────────────────────────────┼────────────┤
│ token_ids (InputLayer)       │ (None, None)                           │          0 │
├──────────────────────────────┼────────────────────────────────────────┼────────────┤
│ bert_backbone (BertBackbone) │ {sequence_output: (None, None, 128),   │  4,385,920 │
│                              │ pooled_output: (None, 128)}            │            │
├──────────────────────────────┼────────────────────────────────────────┼────────────┤
│ dropout (Dropout)            │ (None, 128)                            │          0 │
├──────────────────────────────┼────────────────────────────────────────┼────────────┤
│ logits (Dense)               │ (None, 2)                              │        258 │
└──────────────────────────────┴────────────────────────────────────────┴────────────┘
 Total params: 4,386,178 (16.73 MB)
 Trainable params: 4,386,178 (16.73 MB)
 Non-trainable params: 0 (0.00 B)
```

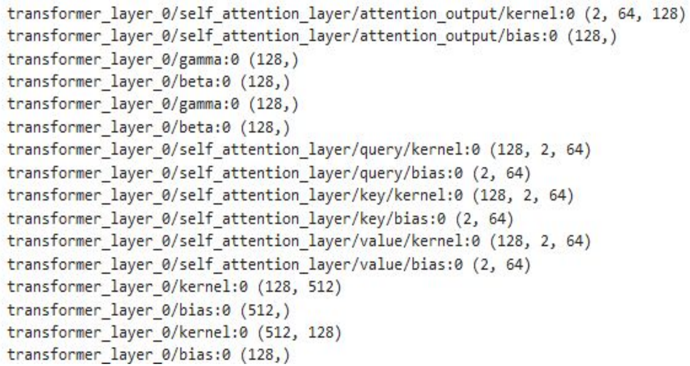


```
Model: "bert_classifier"
┌──────────────────────────────┬────────────────────────────────────────┬────────────┐
│ Layer (type)                 │ Output Shape                           │    Param # │
├──────────────────────────────┼────────────────────────────────────────┼────────────┤
│ padding_mask (InputLayer)    │ (None, None)                           │          0 │
├──────────────────────────────┼────────────────────────────────────────┼────────────┤
│ segment_ids (InputLayer)     │ (None, None)                           │          0 │
├──────────────────────────────┼────────────────────────────────────────┼────────────┤
│ token_ids (InputLayer)       │ (None, None)                           │          0 │
├──────────────────────────────┼────────────────────────────────────────┼────────────┤
│ bert_backbone (BertBackbone) │ {sequence_output: (None, None, 128),   │  4,494,484 │
│                              │ pooled_output: (None, 128)}            │            │
├──────────────────────────────┼────────────────────────────────────────┼────────────┤
│ dropout_22 (Dropout)         │ (None, 128)                            │          0 │
├──────────────────────────────┼────────────────────────────────────────┼────────────┤
│ logits (Dense)               │ (None, 2)                              │        258 │
└──────────────────────────────┴────────────────────────────────────────┴────────────┘
 Total params: 4,494,742 (17.15 MB)
 Trainable params: 142,592 (557.00 KB)
 Non-trainable params: 4,352,150 (16.60 MB)
```

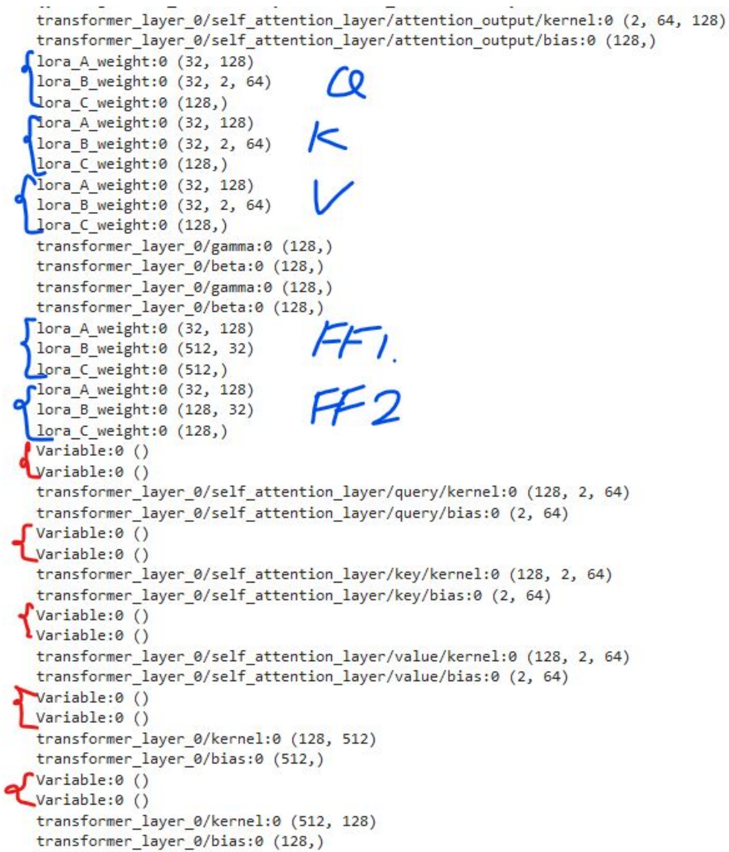


```
Model: "bert_backbone"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 token_ids (InputLayer)      [(None, None)]               0         []

 token_embedding (Reversibl  (None, None, 128)            3906816   ['token_ids[0][0]']
 eEmbedding)

 segment_ids (InputLayer)    [(None, None)]               0         []

 position_embedding (Positi  (None, None, 128)            65536     ['token_embedding[0][0]']
 onEmbedding)

 segment_embedding (Embeddi  (None, None, 128)            256       ['segment_ids[0][0]']
 ng)

 add_22 (Add)                (None, None, 128)            0         ['token_embedding[0][0]',
                                                                     'position_embedding[0][0]',
                                                                     'segment_embedding[0][0]']

 embeddings_layer_norm (Lay  (None, None, 128)            256       ['add_22[0][0]']
 erNormalization)

 embeddings_dropout (Dropou  (None, None, 128)            0         ['embeddings_layer_norm[0][0]'
 t)                                                                 ]

 padding_mask (InputLayer)   [(None, None)]               0         []

 transformer_layer_0 (Trans  (None, None, 128)            252554    ['embeddings_dropout[0][0]',
 formerEncoder)                                                      'padding_mask[0][0]']

 transformer_layer_1 (Trans  (None, None, 128)            252554    ['transformer_layer_0[0][0]',
 formerEncoder)                                                      'padding_mask[0][0]']

 pooled_dense (Dense)        (None, None, 128)            16512     ['transformer_layer_1[0][0]']

 tf.__operators__.getitem_2  (None, 128)                  0         ['pooled_dense[0][0]']
 2 (SlicingOpLambda)

==================================================================================================
Total params: 4494484 (17.15 MB)
Trainable params: 142592 (557.00 KB)
Non-trainable params: 4351892 (16.60 MB)
__________________________________________________________________________________________________
```

Single Encoder Block's weights after Implementing LoRA Layer

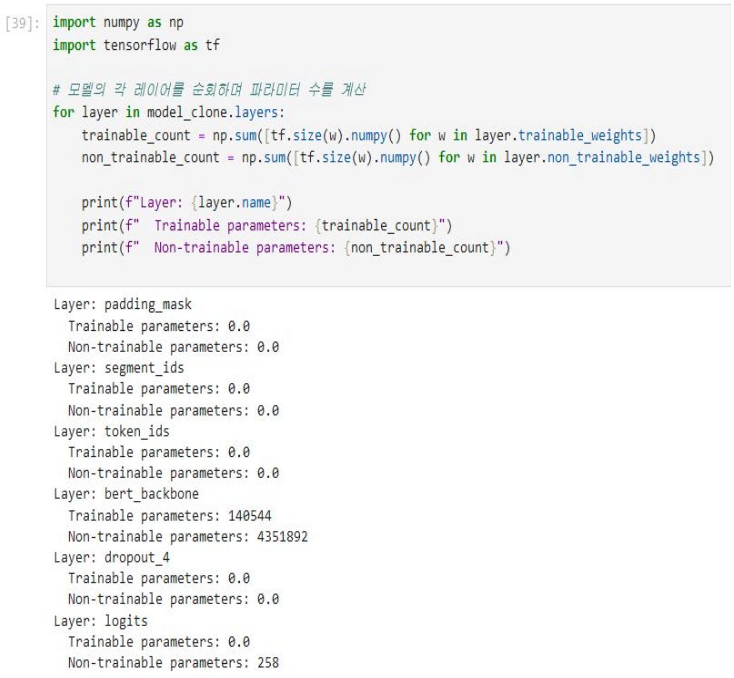


```
transformer_layer_0/self_attention_layer/attention_output/kernel:0 (2, 64, 128)
transformer_layer_0/self_attention_layer/attention_output/bias:0 (128,)
transformer_layer_0/gamma:0 (128,)
transformer_layer_0/beta:0 (128,)
transformer_layer_0/gamma:0 (128,)
transformer_layer_0/beta:0 (128,)
transformer_layer_0/self_attention_layer/query/kernel:0 (128, 2, 64)
transformer_layer_0/self_attention_layer/query/bias:0 (2, 64)
transformer_layer_0/self_attention_layer/key/kernel:0 (128, 2, 64)
transformer_layer_0/self_attention_layer/key/bias:0 (2, 64)
transformer_layer_0/self_attention_layer/value/kernel:0 (128, 2, 64)
transformer_layer_0/self_attention_layer/value/bias:0 (2, 64)
transformer_layer_0/kernel:0 (128, 512)
transformer_layer_0/bias:0 (512,)
transformer_layer_0/kernel:0 (512, 128)
transformer_layer_0/bias:0 (128,)
```



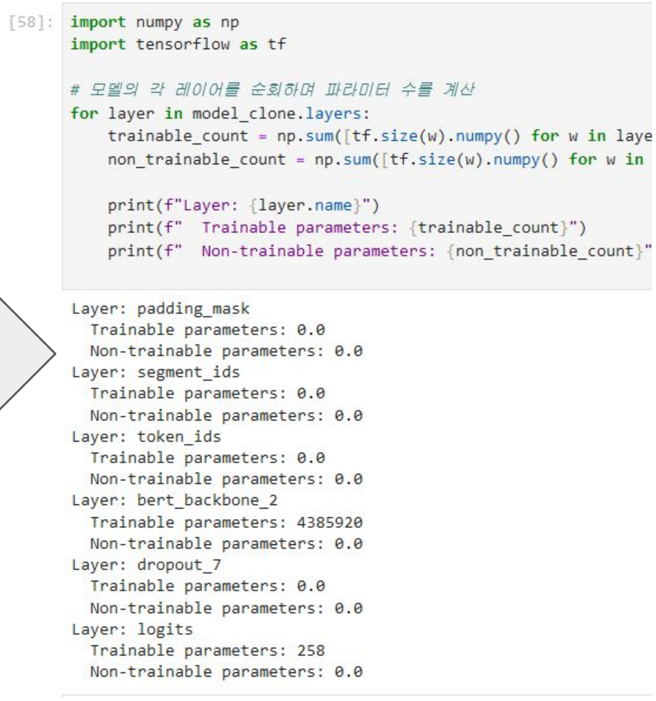
```
transformer_layer_0/self_attention_layer/attention_output/kernel:0 (2, 64, 128)
transformer_layer_0/self_attention_layer/attention_output/bias:0 (128,)
lora_A_weight:0 (32, 128)
lora_B_weight:0 (32, 2, 64)
lora_C_weight:0 (128,)
lora_A_weight:0 (32, 128)
lora_B_weight:0 (32, 2, 64)
lora_C_weight:0 (128,)
lora_A_weight:0 (32, 128)
lora_B_weight:0 (32, 2, 64)
lora_C_weight:0 (128,)
transformer_layer_0/gamma:0 (128,)
transformer_layer_0/beta:0 (128,)
transformer_layer_0/gamma:0 (128,)
transformer_layer_0/beta:0 (128,)
lora_A_weight:0 (32, 128)
lora_B_weight:0 (512, 32)
lora_C_weight:0 (512,)
lora_A_weight:0 (32, 128)
lora_B_weight:0 (128, 32)
lora_C_weight:0 (128,)
Variable:0 ()
Variable:0 ()
transformer_layer_0/self_attention_layer/query/kernel:0 (128, 2, 64)
transformer_layer_0/self_attention_layer/query/bias:0 (2, 64)
Variable:0 ()
Variable:0 ()
transformer_layer_0/self_attention_layer/key/kernel:0 (128, 2, 64)
transformer_layer_0/self_attention_layer/key/bias:0 (2, 64)
Variable:0 ()
Variable:0 ()
transformer_layer_0/self_attention_layer/value/kernel:0 (128, 2, 64)
transformer_layer_0/self_attention_layer/value/bias:0 (2, 64)
Variable:0 ()
Variable:0 ()
transformer_layer_0/kernel:0 (128, 512)
transformer_layer_0/bias:0 (512,)
Variable:0 ()
Variable:0 ()
transformer_layer_0/kernel:0 (512, 128)
transformer_layer_0/bias:0 (128,)
```

의문점 : Compile 후 weight trainable false 했던 것 다시 원복 됨..



```
[39]: import numpy as np
      import tensorflow as tf

      # 모델의 각 레이어를 순회하며 파라미터 수를 계산
      for layer in model_clone.layers:
          trainable_count = np.sum([tf.size(w).numpy() for w in layer.trainable_weights])
          non_trainable_count = np.sum([tf.size(w).numpy() for w in layer.non_trainable_weights])

          print(f"Layer: {layer.name}")
          print(f"  Trainable parameters: {trainable_count}")
          print(f"  Non-trainable parameters: {non_trainable_count}")
```

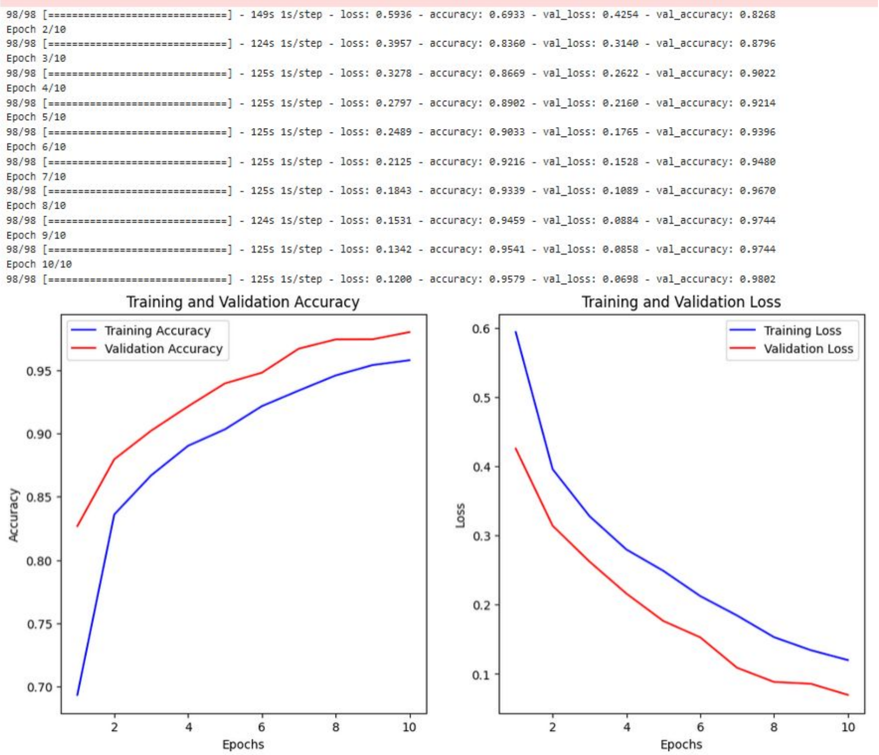
```
Layer: padding_mask
  Trainable parameters: 0.0
  Non-trainable parameters: 0.0
Layer: segment_ids
  Trainable parameters: 0.0
  Non-trainable parameters: 0.0
Layer: token_ids
  Trainable parameters: 0.0
  Non-trainable parameters: 0.0
Layer: bert_backbone
  Trainable parameters: 140544
  Non-trainable parameters: 4351892
Layer: dropout_4
  Trainable parameters: 0.0
  Non-trainable parameters: 0.0
Layer: logits
  Trainable parameters: 0.0
  Non-trainable parameters: 258
```



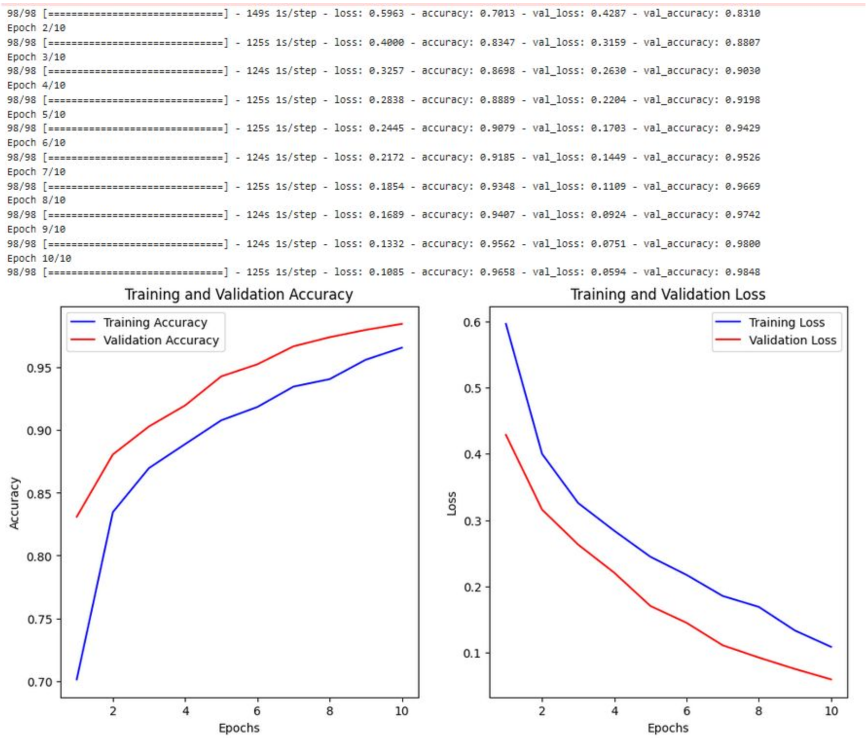
```
[58]: import numpy as np
      import tensorflow as tf

      # 모델의 각 레이어를 순회하며 파라미터 수를 계산
      for layer in model_clone.layers:
          trainable_count = np.sum([tf.size(w).numpy() for w in laye
          non_trainable_count = np.sum([tf.size(w).numpy() for w in

          print(f"Layer: {layer.name}")
          print(f"  Trainable parameters: {trainable_count}")
          print(f"  Non-trainable parameters: {non_trainable_count}"
```

```
Layer: padding_mask
  Trainable parameters: 0.0
  Non-trainable parameters: 0.0
Layer: segment_ids
  Trainable parameters: 0.0
  Non-trainable parameters: 0.0
Layer: token_ids
  Trainable parameters: 0.0
  Non-trainable parameters: 0.0
Layer: bert_backbone_2
  Trainable parameters: 4385920
  Non-trainable parameters: 0.0
Layer: dropout_7
  Trainable parameters: 0.0
  Non-trainable parameters: 0.0
Layer: logits
  Trainable parameters: 258
  Non-trainable parameters: 0.0
```

Just Weight Decay VS Weight Decay with Noise (+C_weights : bias for LoRA)

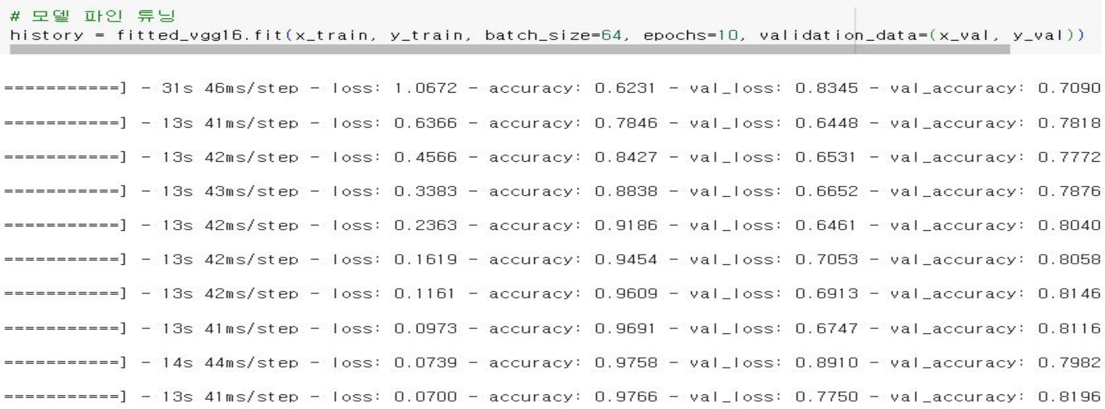


```
98/98 [==============================] - 149s 1s/step - loss: 0.5936 - accuracy: 0.6933 - val_loss: 0.4254 - val_accuracy: 0.8268
Epoch 2/10
98/98 [==============================] - 124s 1s/step - loss: 0.3957 - accuracy: 0.8360 - val_loss: 0.3140 - val_accuracy: 0.8796
Epoch 3/10
98/98 [==============================] - 125s 1s/step - loss: 0.3278 - accuracy: 0.8669 - val_loss: 0.2622 - val_accuracy: 0.9022
Epoch 4/10
98/98 [==============================] - 125s 1s/step - loss: 0.2797 - accuracy: 0.8902 - val_loss: 0.2160 - val_accuracy: 0.9214
Epoch 5/10
98/98 [==============================] - 125s 1s/step - loss: 0.2489 - accuracy: 0.9033 - val_loss: 0.1765 - val_accuracy: 0.9396
Epoch 6/10
98/98 [==============================] - 125s 1s/step - loss: 0.2125 - accuracy: 0.9216 - val_loss: 0.1528 - val_accuracy: 0.9480
Epoch 7/10
98/98 [==============================] - 125s 1s/step - loss: 0.1843 - accuracy: 0.9339 - val_loss: 0.1089 - val_accuracy: 0.9670
Epoch 8/10
98/98 [==============================] - 124s 1s/step - loss: 0.1531 - accuracy: 0.9459 - val_loss: 0.0884 - val_accuracy: 0.9744
Epoch 9/10
98/98 [==============================] - 125s 1s/step - loss: 0.1342 - accuracy: 0.9541 - val_loss: 0.0858 - val_accuracy: 0.9744
Epoch 10/10
98/98 [==============================] - 125s 1s/step - loss: 0.1200 - accuracy: 0.9579 - val_loss: 0.0698 - val_accuracy: 0.9802
```



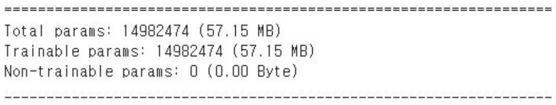
```
98/98 [==============================] - 149s 1s/step - loss: 0.5963 - accuracy: 0.7013 - val_loss: 0.4287 - val_accuracy: 0.8310
Epoch 2/10
98/98 [==============================] - 125s 1s/step - loss: 0.4000 - accuracy: 0.8347 - val_loss: 0.3159 - val_accuracy: 0.8807
Epoch 3/10
98/98 [==============================] - 124s 1s/step - loss: 0.3257 - accuracy: 0.8698 - val_loss: 0.2630 - val_accuracy: 0.9030
Epoch 4/10
98/98 [==============================] - 125s 1s/step - loss: 0.2838 - accuracy: 0.8889 - val_loss: 0.2204 - val_accuracy: 0.9198
Epoch 5/10
98/98 [==============================] - 125s 1s/step - loss: 0.2445 - accuracy: 0.9079 - val_loss: 0.1703 - val_accuracy: 0.9429
Epoch 6/10
98/98 [==============================] - 124s 1s/step - loss: 0.2172 - accuracy: 0.9185 - val_loss: 0.1449 - val_accuracy: 0.9526
Epoch 7/10
98/98 [==============================] - 125s 1s/step - loss: 0.1854 - accuracy: 0.9348 - val_loss: 0.1109 - val_accuracy: 0.9669
Epoch 8/10
98/98 [==============================] - 124s 1s/step - loss: 0.1689 - accuracy: 0.9407 - val_loss: 0.0924 - val_accuracy: 0.9742
Epoch 9/10
98/98 [==============================] - 124s 1s/step - loss: 0.1332 - accuracy: 0.9562 - val_loss: 0.0751 - val_accuracy: 0.9800
Epoch 10/10
98/98 [==============================] - 125s 1s/step - loss: 0.1085 - accuracy: 0.9658 - val_loss: 0.0594 - val_accuracy: 0.9848
```

scheduling factor 적용 전, convlora만 적용된 vgg16

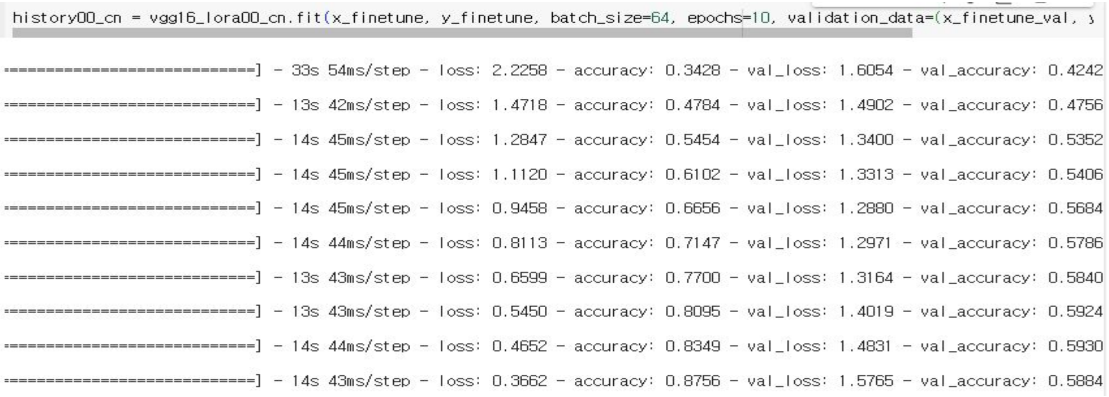


```
# 모델 파인 튜닝
history = fitted_vgg16.fit(x_train, y_train, batch_size=64, epochs=10, validation_data=(x_val, y_val))
```

```
===========] - 31s 46ms/step - loss: 1.0672 - accuracy: 0.6231 - val_loss: 0.8345 - val_accuracy: 0.7090
===========] - 13s 41ms/step - loss: 0.6366 - accuracy: 0.7846 - val_loss: 0.6448 - val_accuracy: 0.7818
===========] - 13s 42ms/step - loss: 0.4566 - accuracy: 0.8427 - val_loss: 0.6531 - val_accuracy: 0.7772
===========] - 13s 43ms/step - loss: 0.3383 - accuracy: 0.8838 - val_loss: 0.6652 - val_accuracy: 0.7876
===========] - 13s 42ms/step - loss: 0.2363 - accuracy: 0.9186 - val_loss: 0.6461 - val_accuracy: 0.8040
===========] - 13s 42ms/step - loss: 0.1619 - accuracy: 0.9454 - val_loss: 0.7053 - val_accuracy: 0.8058
===========] - 13s 42ms/step - loss: 0.1161 - accuracy: 0.9609 - val_loss: 0.6913 - val_accuracy: 0.8146
===========] - 13s 41ms/step - loss: 0.0973 - accuracy: 0.9691 - val_loss: 0.6747 - val_accuracy: 0.8116
===========] - 14s 44ms/step - loss: 0.0739 - accuracy: 0.9758 - val_loss: 0.8910 - val_accuracy: 0.7982
===========] - 13s 41ms/step - loss: 0.0700 - accuracy: 0.9766 - val_loss: 0.7750 - val_accuracy: 0.8196
```

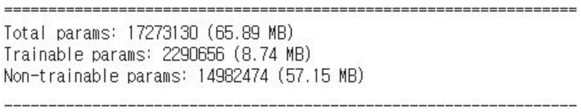


```
=================================================================
Total params: 14982474 (57.15 MB)
Trainable params: 14982474 (57.15 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



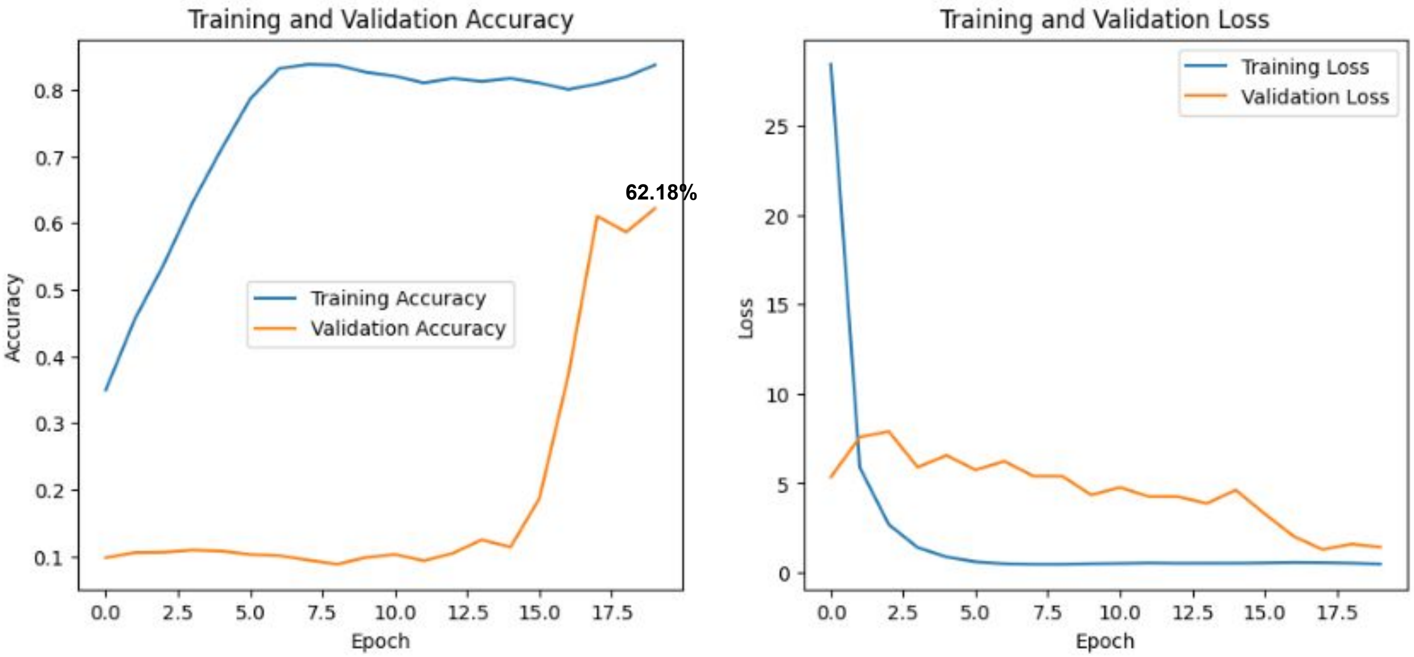
```
history00_cn = vgg16_lora00_cn.fit(x_finetune, y_finetune, batch_size=64, epochs=10, validation_data=(x_finetune_val, y
```

```
===========================] - 33s 54ms/step - loss: 2.2258 - accuracy: 0.3428 - val_loss: 1.6054 - val_accuracy: 0.4242
===========================] - 13s 42ms/step - loss: 1.4718 - accuracy: 0.4784 - val_loss: 1.4902 - val_accuracy: 0.4756
===========================] - 14s 45ms/step - loss: 1.2847 - accuracy: 0.5454 - val_loss: 1.3400 - val_accuracy: 0.5352
===========================] - 14s 45ms/step - loss: 1.1120 - accuracy: 0.6102 - val_loss: 1.3313 - val_accuracy: 0.5406
===========================] - 14s 45ms/step - loss: 0.9458 - accuracy: 0.6656 - val_loss: 1.2880 - val_accuracy: 0.5684
===========================] - 14s 44ms/step - loss: 0.8113 - accuracy: 0.7147 - val_loss: 1.2971 - val_accuracy: 0.5786
===========================] - 13s 43ms/step - loss: 0.6599 - accuracy: 0.7700 - val_loss: 1.3164 - val_accuracy: 0.5840
===========================] - 13s 43ms/step - loss: 0.5450 - accuracy: 0.8095 - val_loss: 1.4019 - val_accuracy: 0.5924
===========================] - 14s 44ms/step - loss: 0.4652 - accuracy: 0.8349 - val_loss: 1.4831 - val_accuracy: 0.5930
===========================] - 14s 43ms/step - loss: 0.3662 - accuracy: 0.8756 - val_loss: 1.5765 - val_accuracy: 0.5884
```



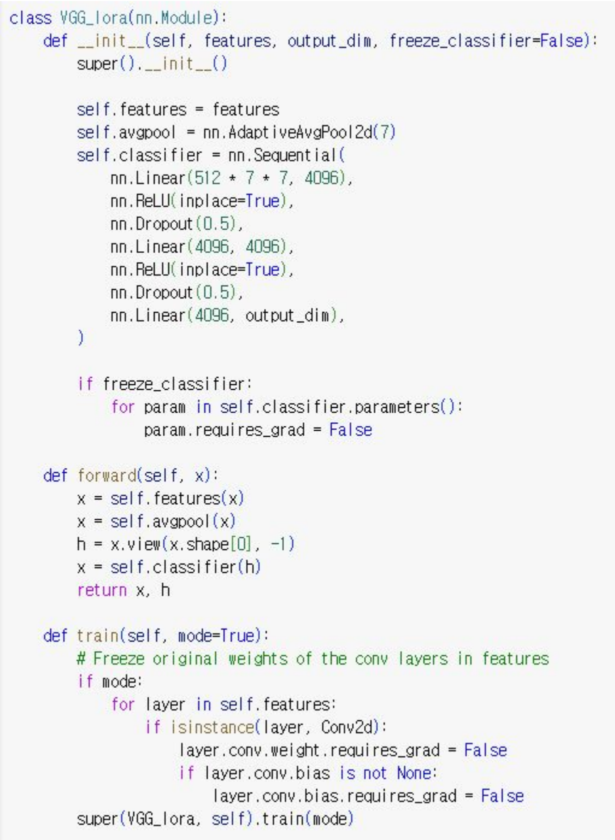
```
=================================================================
Total params: 17273130 (65.89 MB)
Trainable params: 2290656 (8.74 MB)
Non-trainable params: 14982474 (57.15 MB)
_________________________________________________________________
```

scheduling factor 적용한 VGG16

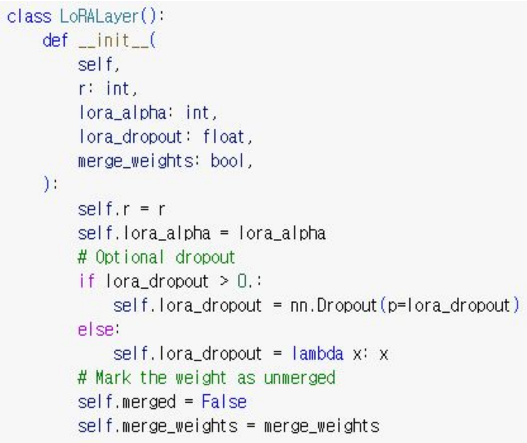


62.18%

ConvLoRA in PyTorch

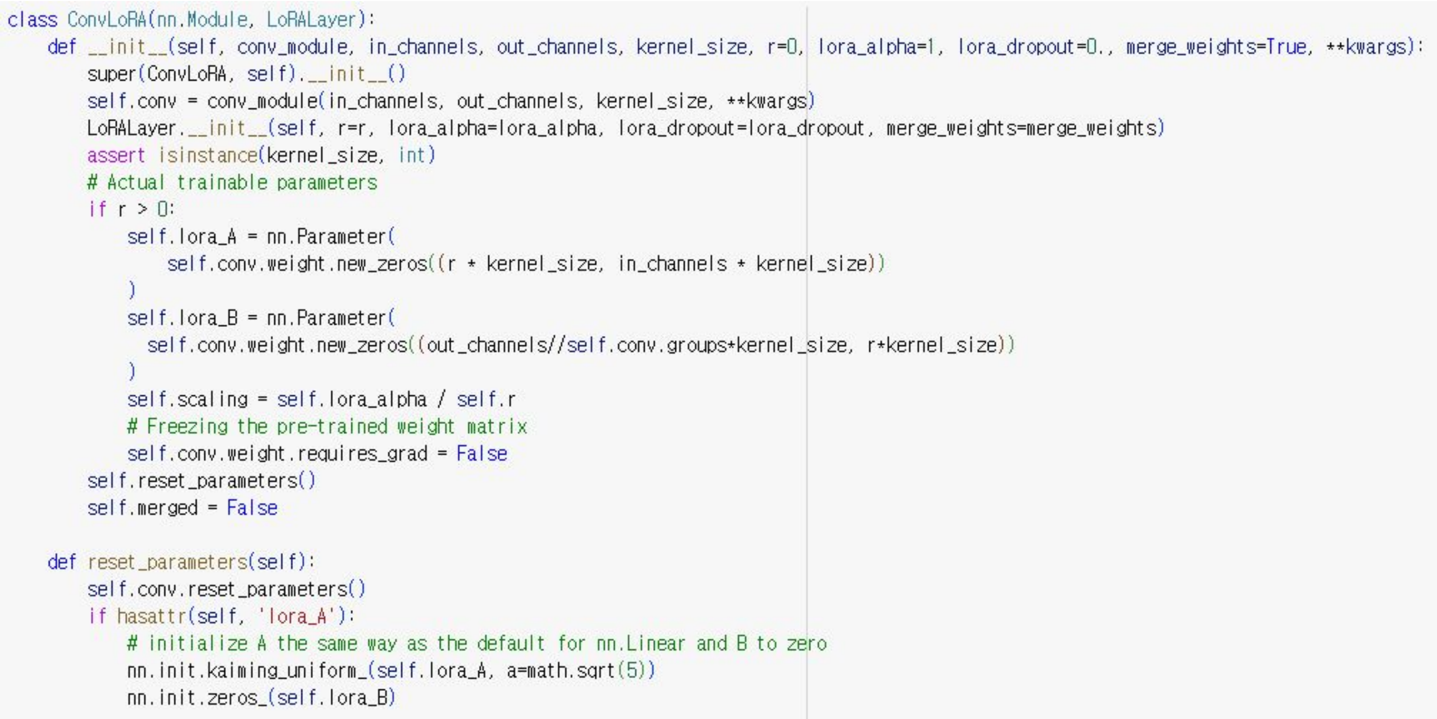


```python
class VGG_lora(nn.Module):
    def __init__(self, features, output_dim, freeze_classifier=False):
        super().__init__()

        self.features = features
        self.avgpool = nn.AdaptiveAvgPool2d(7)
        self.classifier = nn.Sequential(
            nn.Linear(512 * 7 * 7, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(4096, output_dim),
        )

        if freeze_classifier:
            for param in self.classifier.parameters():
                param.requires_grad = False

    def forward(self, x):
        x = self.features(x)
        x = self.avgpool(x)
        h = x.view(x.shape[0], -1)
        x = self.classifier(h)
        return x, h

    def train(self, mode=True):
        # Freeze original weights of the conv layers in features
        if mode:
            for layer in self.features:
                if isinstance(layer, Conv2d):
                    layer.conv.weight.requires_grad = False
                    if layer.conv.bias is not None:
                        layer.conv.bias.requires_grad = False
        super(VGG_lora, self).train(mode)
```

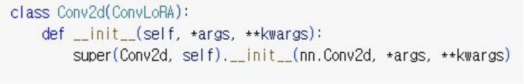


```python
class LoRALayer():
    def __init__(
        self,
        r: int,
        lora_alpha: int,
        lora_dropout: float,
        merge_weights: bool,
    ):
        self.r = r
        self.lora_alpha = lora_alpha
        # Optional dropout
        if lora_dropout > 0.:
            self.lora_dropout = nn.Dropout(p=lora_dropout)
        else:
            self.lora_dropout = lambda x: x
        # Mark the weight as unmerged
        self.merged = False
        self.merge_weights = merge_weights
```

ConvLoRA in PyTorch

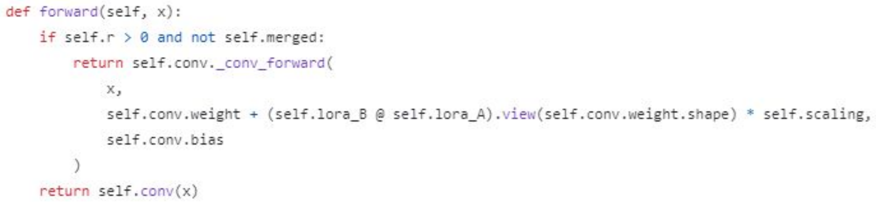


```
class ConvLoRA(nn.Module, LoRALayer):
    def __init__(self, conv_module, in_channels, out_channels, kernel_size, r=0, lora_alpha=1, lora_dropout=0., merge_weights=True, **kwargs):
        super(ConvLoRA, self).__init__()
        self.conv = conv_module(in_channels, out_channels, kernel_size, **kwargs)
        LoRALayer.__init__(self, r=r, lora_alpha=lora_alpha, lora_dropout=lora_dropout, merge_weights=merge_weights)
        assert isinstance(kernel_size, int)
        # Actual trainable parameters
        if r > 0:
            self.lora_A = nn.Parameter(
                self.conv.weight.new_zeros((r * kernel_size, in_channels * kernel_size))
            )
            self.lora_B = nn.Parameter(
              self.conv.weight.new_zeros((out_channels//self.conv.groups*kernel_size, r*kernel_size))
            )
            self.scaling = self.lora_alpha / self.r
            # Freezing the pre-trained weight matrix
            self.conv.weight.requires_grad = False
        self.reset_parameters()
        self.merged = False

    def reset_parameters(self):
        self.conv.reset_parameters()
        if hasattr(self, 'lora_A'):
            # initialize A the same way as the default for nn.Linear and B to zero
            nn.init.kaiming_uniform_(self.lora_A, a=math.sqrt(5))
            nn.init.zeros_(self.lora_B)
```

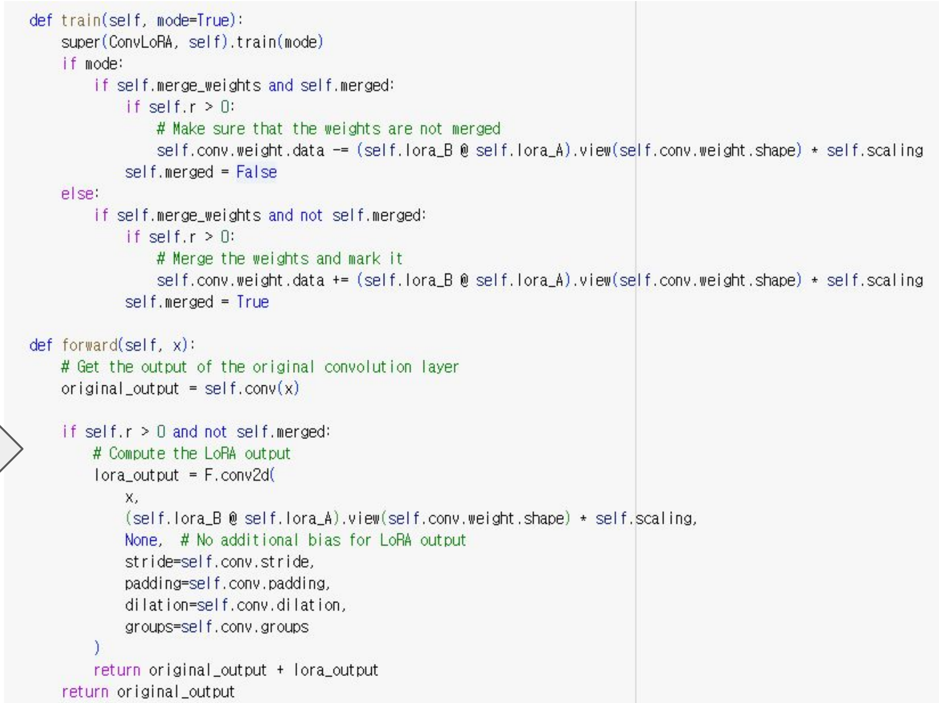


```
class Conv2d(ConvLoRA):
    def __init__(self, *args, **kwargs):
        super(Conv2d, self).__init__(nn.Conv2d, *args, **kwargs)
```

ConvLoRA in PyTorch

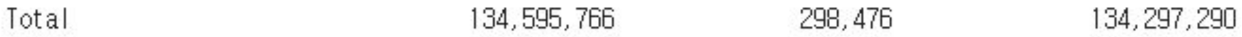


```python
def forward(self, x):
    if self.r > 0 and not self.merged:
        return self.conv._conv_forward(
            x,
            self.conv.weight + (self.lora_B @ self.lora_A).view(self.conv.weight.shape) * self.scaling,
            self.conv.bias
        )
    return self.conv(x)
```

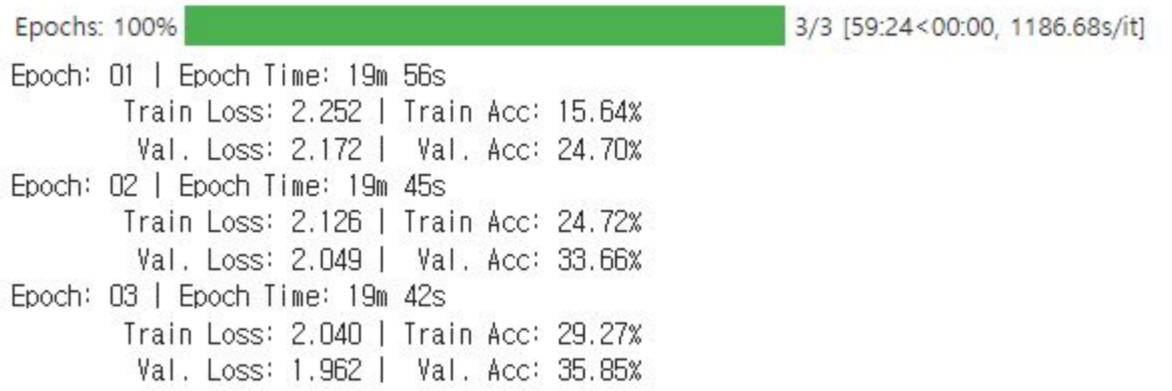


```python
def train(self, mode=True):
    super(ConvLoRA, self).train(mode)
    if mode:
        if self.merge_weights and self.merged:
            if self.r > 0:
                # Make sure that the weights are not merged
                self.conv.weight.data -= (self.lora_B @ self.lora_A).view(self.conv.weight.shape) * self.scaling
            self.merged = False
    else:
        if self.merge_weights and not self.merged:
            if self.r > 0:
                # Merge the weights and mark it
                self.conv.weight.data += (self.lora_B @ self.lora_A).view(self.conv.weight.shape) * self.scaling
            self.merged = True

def forward(self, x):
    # Get the output of the original convolution layer
    original_output = self.conv(x)

    if self.r > 0 and not self.merged:
        # Compute the LoRA output
        lora_output = F.conv2d(
            x,
            (self.lora_B @ self.lora_A).view(self.conv.weight.shape) * self.scaling,
            None,  # No additional bias for LoRA output
            stride=self.conv.stride,
            padding=self.conv.padding,
            dilation=self.conv.dilation,
            groups=self.conv.groups
        )
        return original_output + lora_output
    return original_output
```

ConvLoRA in PyTorch



```
Total                          134,595,766          298,476          134,297,290
```

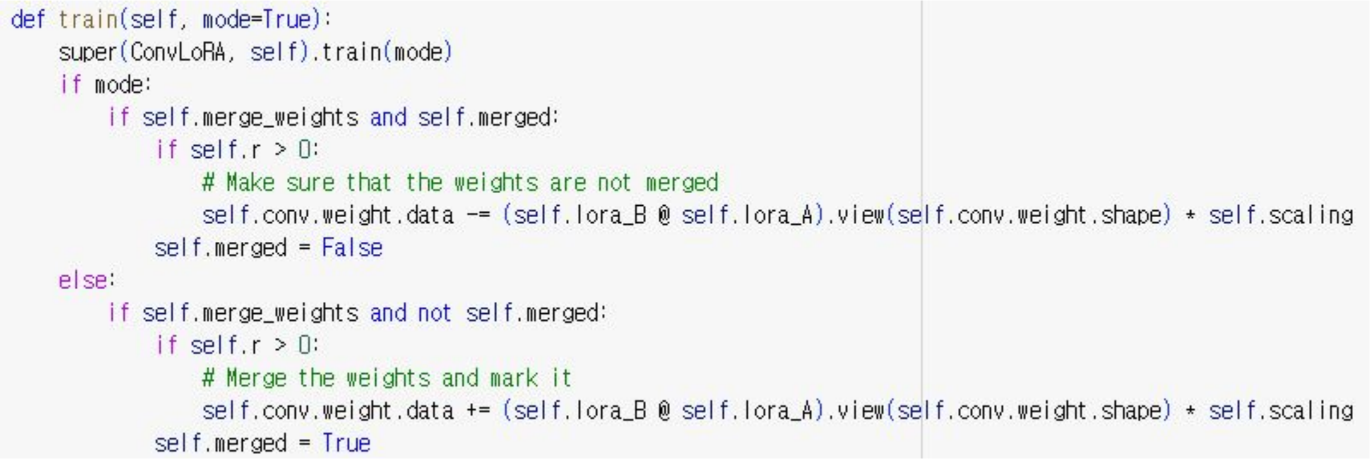


```
Epochs: 100% ████████████████████ 3/3 [59:24<00:00, 1186.68s/it]
Epoch: 01 | Epoch Time: 19m 56s
	Train Loss: 2.252 | Train Acc: 15.64%
	 Val. Loss: 2.172 |  Val. Acc: 24.70%
Epoch: 02 | Epoch Time: 19m 45s
	Train Loss: 2.126 | Train Acc: 24.72%
	 Val. Loss: 2.049 |  Val. Acc: 33.66%
Epoch: 03 | Epoch Time: 19m 42s
	Train Loss: 2.040 | Train Acc: 29.27%
	 Val. Loss: 1.962 |  Val. Acc: 35.85%
```

ConvLoRA in PyTorch-Issues

ConvLora class의 train

- mode가 아닌 if trainable, else 형태로 변경할 수 있는지 알아보기

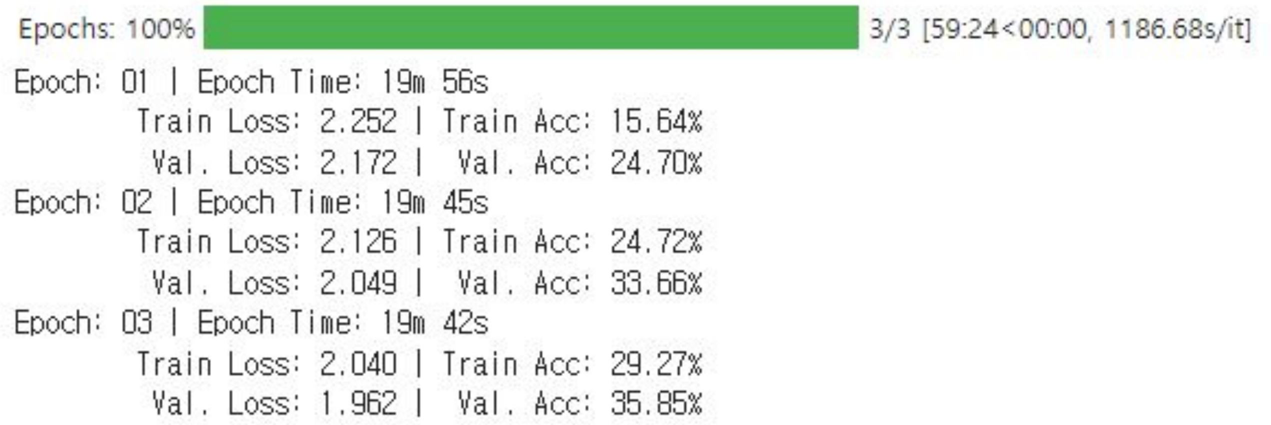


```python
def train(self, mode=True):
    super(ConvLoRA, self).train(mode)
    if mode:
        if self.merge_weights and self.merged:
            if self.r > 0:
                # Make sure that the weights are not merged
                self.conv.weight.data -= (self.lora_B @ self.lora_A).view(self.conv.weight.shape) * self.scaling
            self.merged = False
    else:
        if self.merge_weights and not self.merged:
            if self.r > 0:
                # Merge the weights and mark it
                self.conv.weight.data += (self.lora_B @ self.lora_A).view(self.conv.weight.shape) * self.scaling
            self.merged = True
```

ConvLoRA in PyTorch-Issues

Training time

- 시간 소요가 과하게 되는데 param을 잘못 가져온 것인지 그 외의 문제인지 확인하기



```
Epochs: 100%  3/3 [59:24<00:00, 1186.68s/it]
Epoch: 01 | Epoch Time: 19m 56s
        Train Loss: 2.252 | Train Acc: 15.64%
         Val. Loss: 2.172 |  Val. Acc: 24.70%
Epoch: 02 | Epoch Time: 19m 45s
        Train Loss: 2.126 | Train Acc: 24.72%
         Val. Loss: 2.049 |  Val. Acc: 33.66%
Epoch: 03 | Epoch Time: 19m 42s
        Train Loss: 2.040 | Train Acc: 29.27%
         Val. Loss: 1.962 |  Val. Acc: 35.85%
```