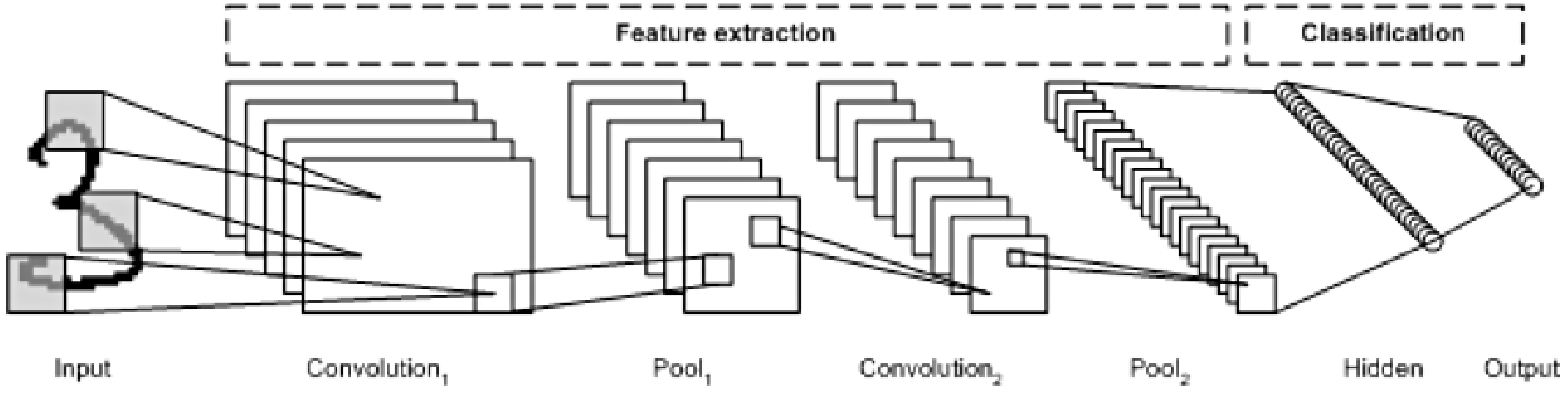
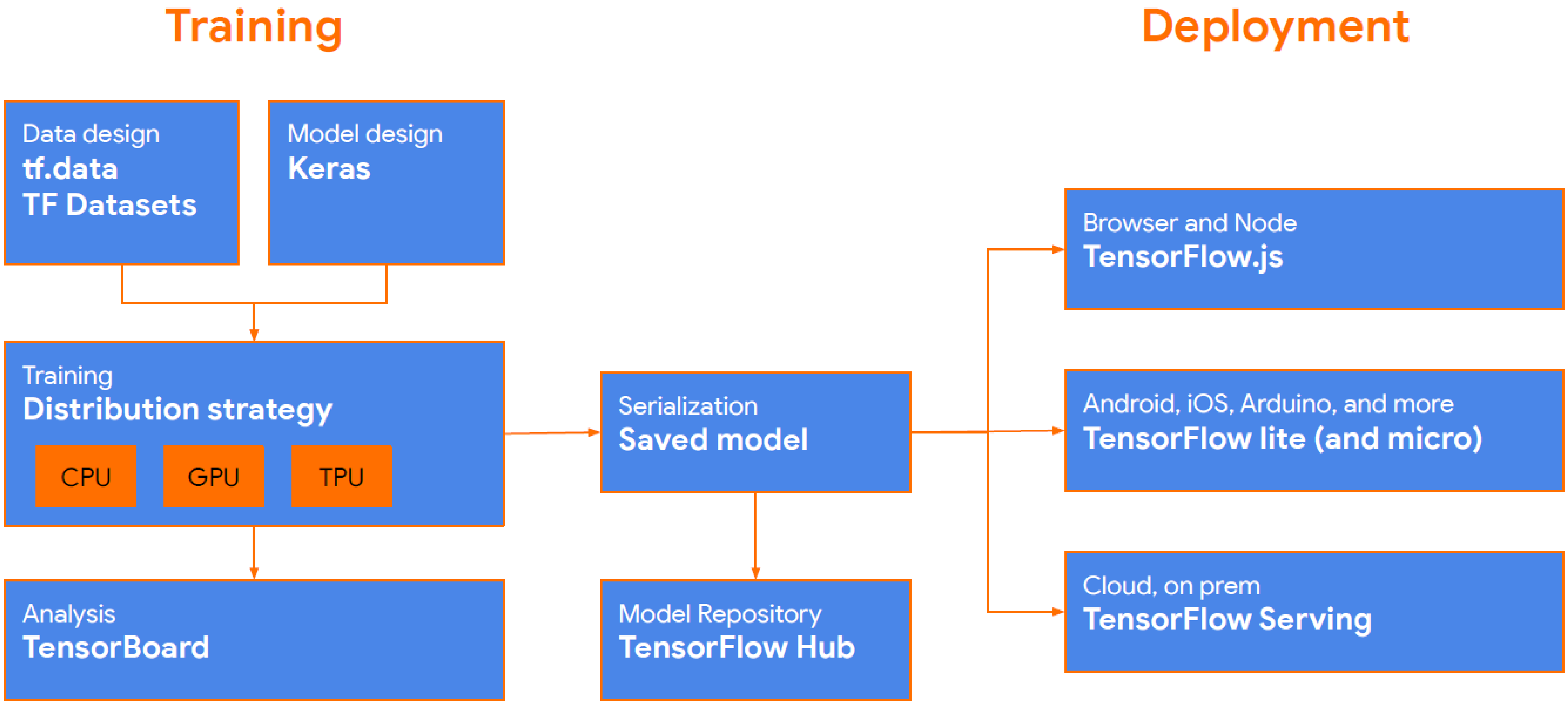
모델 서브클래싱



Feature extraction
Classification
Input
Convolution$_1$
Pool$_1$
Convolution$_2$
Pool$_2$
Hidden
Output

박경규



https://www.youtube.com/watch?v=5ECD8J3dvDQ

Model building

Sequential API
+ built-in layers

New users,
simple models

Functional API
+ built-in layers

Engineers with
standard use
cases

Functional API
+ Custom layers
+ Custom metrics
+ Custom losses

Engineers
requiring
increasing
control

Subclassing: write
everything yourself
from scratch

Researchers

https://www.youtube.com/watch?v=5ECD8J3dvDQ

```
model = keras.Sequential()
model.add(layers.Dense(32, activation='relu', input_shape=(784,)))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(32, activation='softmax'))
```

https://www.youtube.com/watch?v=5ECD8J3dvDQ

Model building

Sequential API
+ built-in layers

New users, simple models

Functional API
+ built-in layers

Engineers with standard use cases

Functional API
+ Custom layers
+ Custom metrics
+ Custom losses

Engineers requiring increasing control

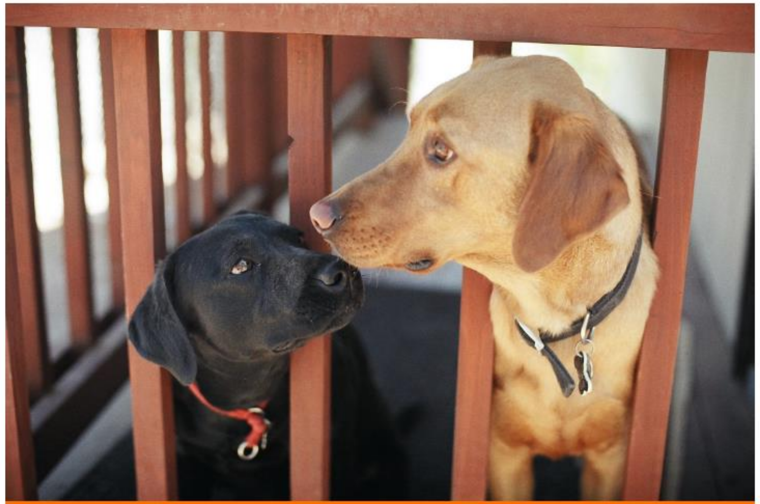
Subclassing: write everything yourself from scratch

Researchers

https://www.youtube.com/watch?v=5ECD8J3dvDQ

Model building

Visual Question Answering



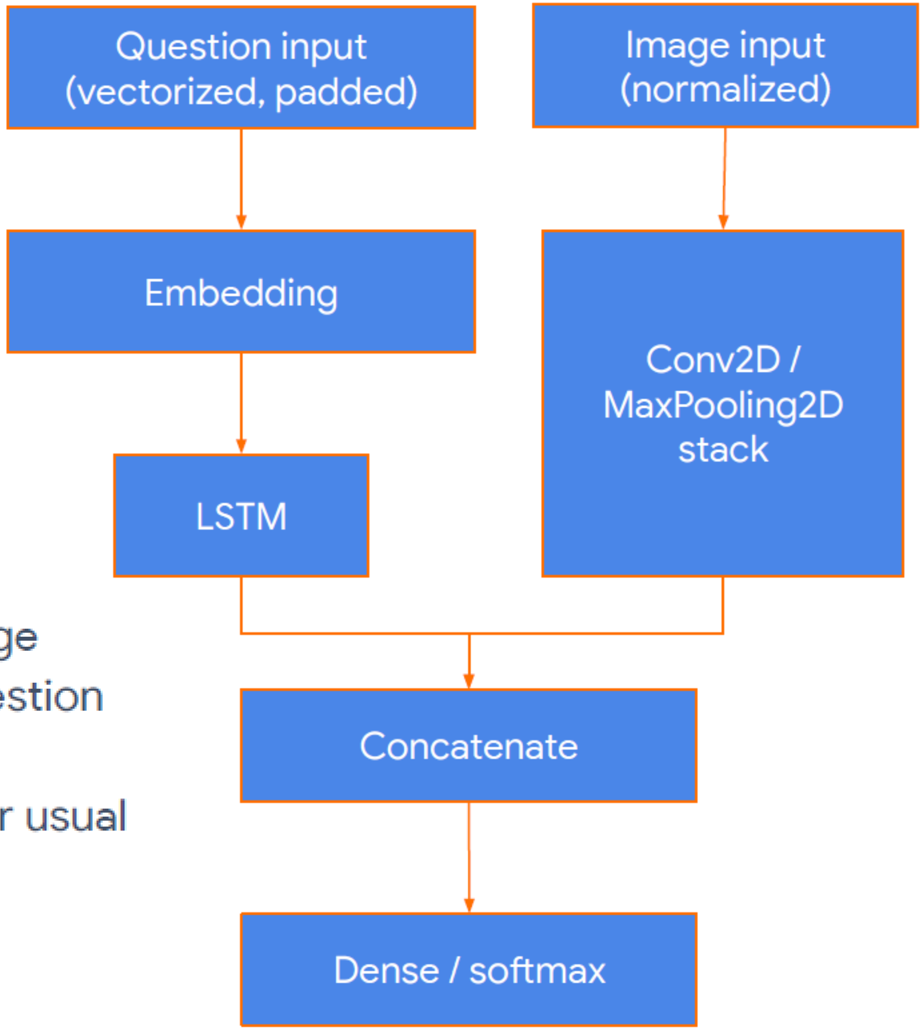
Question: What color is the dog on the right?

Answer: Golden

Workflow

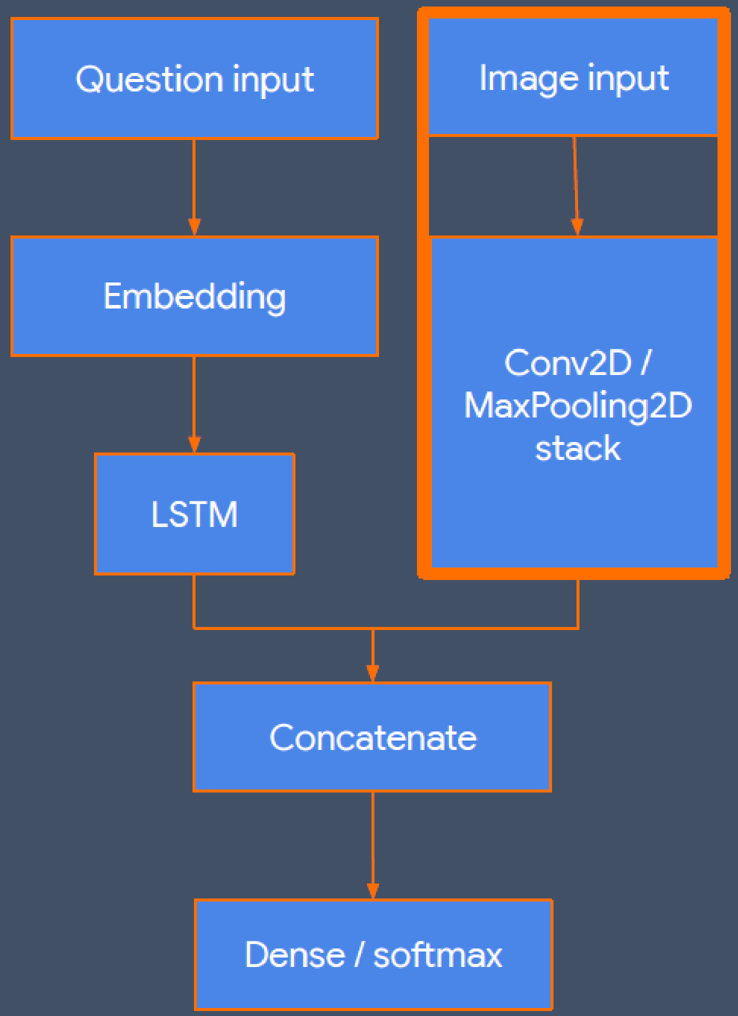
A multi-input model

1. Use a CNN to embed the image
2. Use a LSTM to embed the question
3. **Concatenate**
4. Classify with Dense layers, per usual



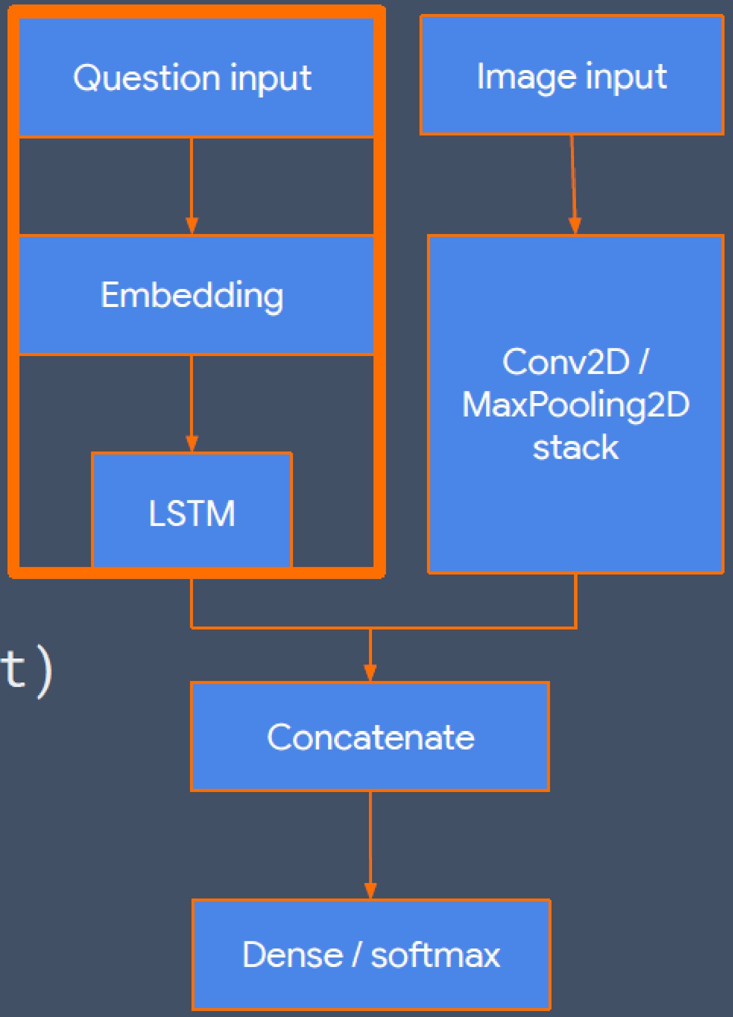
https://www.youtube.com/watch?v=5ECD8J3dvDQ

```python
# A vision model.
# Encode an image into a vector.
vision_model = Sequential()
vision_model.add(Conv2D(64, (3, 3),
                        activation='relu',
                        input_shape=(224, 224, 3)))
vision_model.add(MaxPooling2D())
vision_model.add(Flatten())

# Get a tensor with the output of your vision model
image_input = Input(shape=(224, 224, 3))
encoded_image = vision_model(image_input)
```



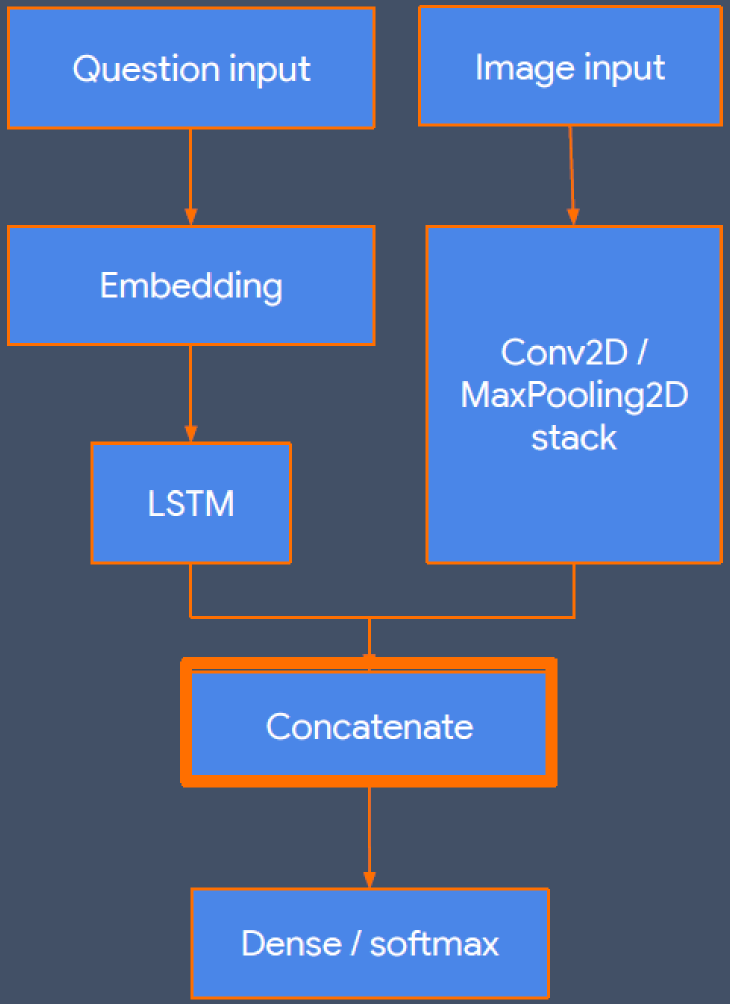
https://www.youtube.com/watch?v=5ECD8J3dvDQ

```python
# A language model.
# Encode the question into a vector.
question_input = Input(shape=(100,),
                       dtype='int32',
                       name="Question")

embedded = Embedding(input_dim=10000,
                     output_dim=256,
                     input_length=100)(question_input)

encoded_question = LSTM(256)(embedded_question)
```



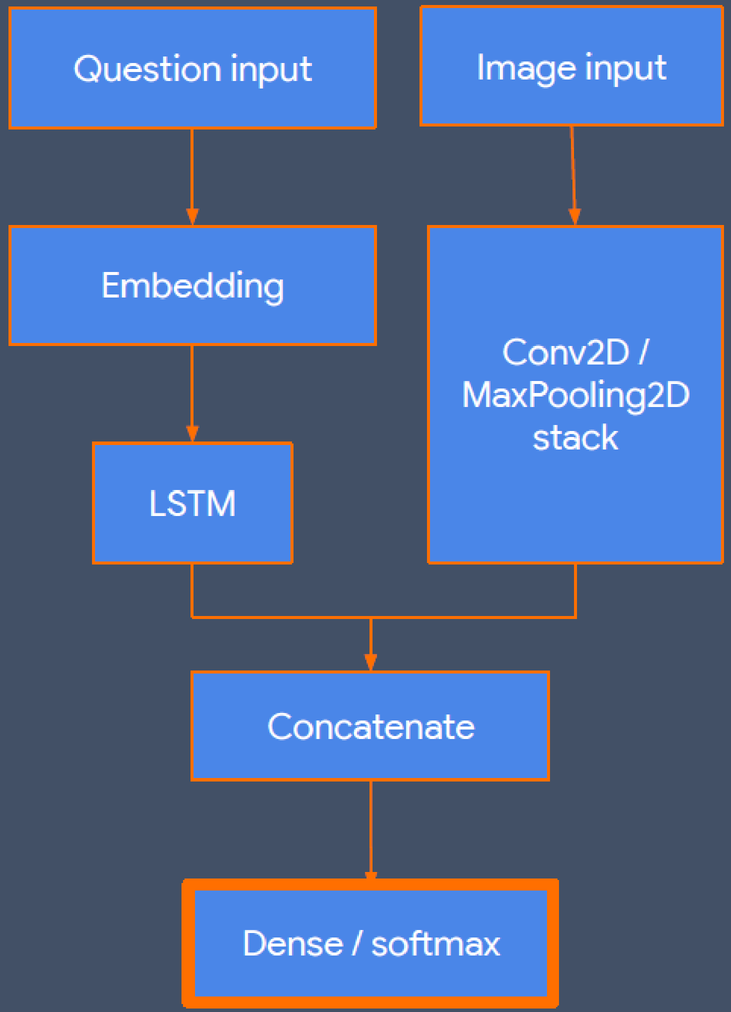
https://www.youtube.com/watch?v=5ECD8J3dvDQ

```
# Concatenate the encoded image and question
merged = layers.concatenate([encoded_image,
                             encoded_question])
```



https://www.youtube.com/watch?v=5ECD8J3dvDQ

```
# Train a classifier on top.
output = Dense(1000,
               activation='softmax')(merged)


# You can train w/ .fit, .train_on_batch,
# or with a GradientTape.
vqa_model = Model(inputs=[image_input,
                          question_input],
                          outputs=output)
```



https://www.youtube.com/watch?v=5ECD8J3dvDQ

Model building

Sequential API
+ built-in layers

New users, simple models

Functional API
+ built-in layers

Engineers with standard use cases

Functional API
- + Custom layers
- + Custom metrics
- + Custom losses

Engineers requiring increasing control

Subclassing: write everything yourself from scratch

Researchers

https://www.youtube.com/watch?v=5ECD8J3dvDQ

```
class MyModel(tf.keras.Model):
  def __init__(self, num_classes=10):
    super(MyModel, self).__init__(name='my_model')
    self.dense_1 = layers.Dense(32, activation='relu')
    self.dense_2 = layers.Dense(num_classes,activation='softmax')

  def call(self, inputs):
    # Define your forward pass here
    x = self.dense_1(inputs)
    return self.dense_2(x)
```


https://www.youtube.com/watch?v=5ECD8J3dvDQ

```python
class MyModel(tf.keras.Model):
  def __init__(self, num_classes=10):
    super(MyModel, self).__init__(name='my_model')
    self.dense_1 = layers.Dense(32)
    self.dense_2 = layers.Dense(num_classes,activation='softmax')

  def call(self, inputs):
    # Define your forward pass here
    x = self.dense_1(inputs)
    x = tf.nn.relu(x)
    return self.dense_2(x)
```

Model training

model.fit()

Quick experiment

model.fit()
+callbacks

Customize your training loop
Add checkpointing, early stopping, TensorBoard monitoring, send Slack notifications...

model.train_on_batch(),
+ callbacks

Custom training loop using built-in optimizers and losses
e.g. GANs

Custom training loop
with GradientTape

Complete control
e.g. new optimization algorithm; easily modify gradients as you go.

https://www.youtube.com/watch?v=5ECD8J3dvDQ

```
model.compile(optimizer=Adam(),
              loss=BinaryCrossentropy(),
              metrics=[AUC(), Precision(), Recall()])

model.fit(data,
          epochs=10,
          validation_data=val_data,
          callbacks=[EarlyStopping(),
                     TensorBoard(),
                     ModelCheckpoint()])
```

...or write your own callbacks!

https://www.youtube.com/watch?v=5ECD8J3dvDQ

Model training

model.fit()

Quick experiment

model.fit() +callbacks

Customize your training loop
Add checkpointing, early stopping, TensorBoard monitoring, ...

model.train_on_batch(), + callbacks

Custom training loop using built-in optimizers and losses
e.g. GANs

Custom training loop with GradientTape

Complete control
e.g. new optimization algorithm; easily modify gradients as you go.

https://www.youtube.com/watch?v=5ECD8J3dvDQ

Graphs with one LOC

```
@tf.function
def train_step(features, labels):
  with tf.GradientTape() as tape:
    logits = model(features, training=True)
    loss = loss_fn(labels, logits)

  grads = tape.gradient(loss, model.trainable_variables)
  optimizer.apply_gradients(zip(grads, model.trainable_variables))
  return loss
```

Gradients

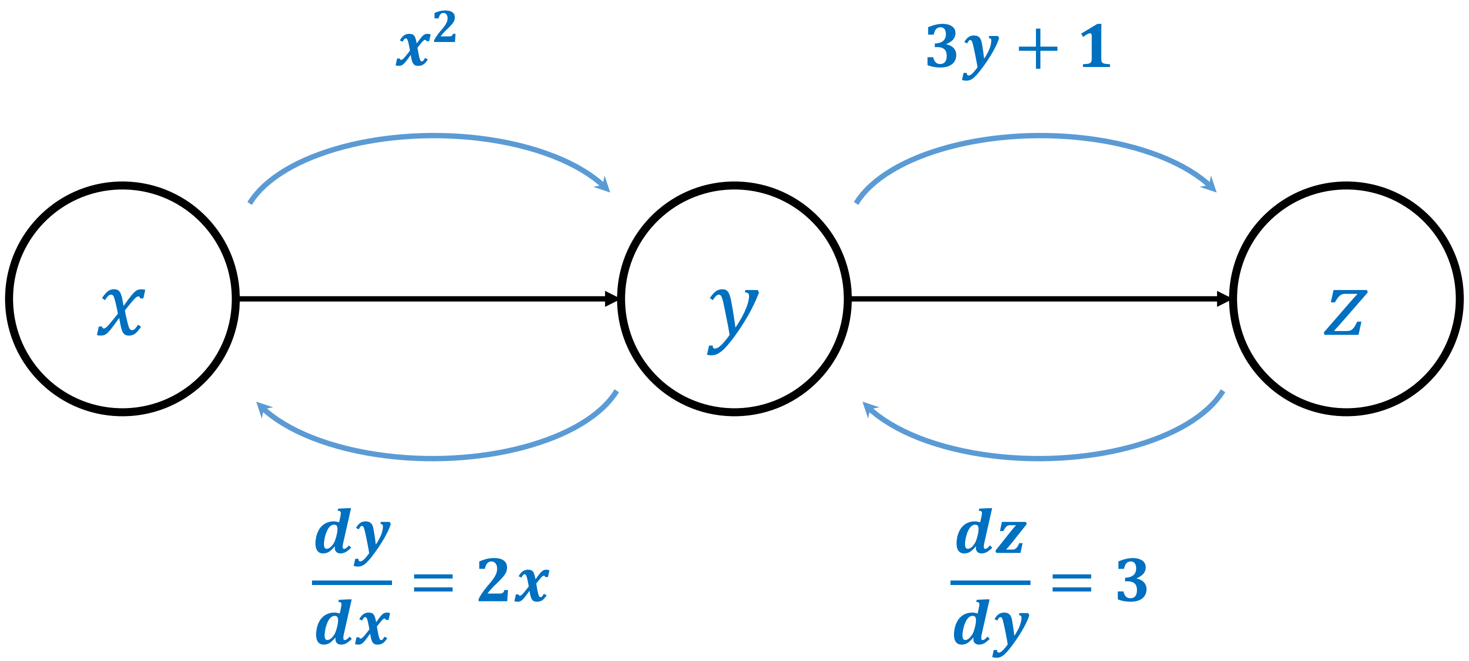
```
x = tf.constant(3.0)
with tf.GradientTape() as g:
  g.watch(x)
  y = x * x
dy_dx = g.gradient(y, x) # 6.0
```

Variables are automatically tracked

```
dense1 = tf.keras.layers.Dense(32)
dense2 = tf.keras.layers.Dense(32)



with tf.GradientTape() as tape:
  result = dense2(dense1(tf.zeros([1, 10])))
  tape.gradient(result, dense1.variables)
```

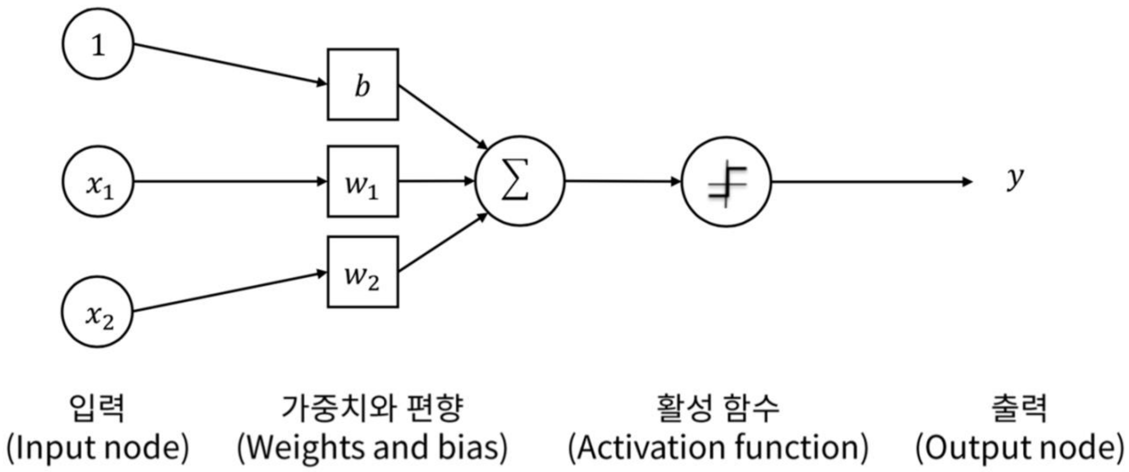
As we saw previously, you can get the gradients for model.trainable_variables as well in one go.

계산그래프(Computational Graph)



Gradient 계산

- 텐서플로는 자동미분(automatic difference)을 지원합니다.
- 중첩된 함수의 그레이디언트를 계산하기 위해 연쇄법칙(Chain Rule)을 구현한 것으로 생각할 수 있습니다.
- 텐서플로는 계산그래프안에서 Tensor의 Gradient를 계산하는 기능을 제공합니다.
- Gradient 를 계산하려면 tf.GradientTape을 통해 계산을 기록해야 합니다.



$$z = wx + b$$

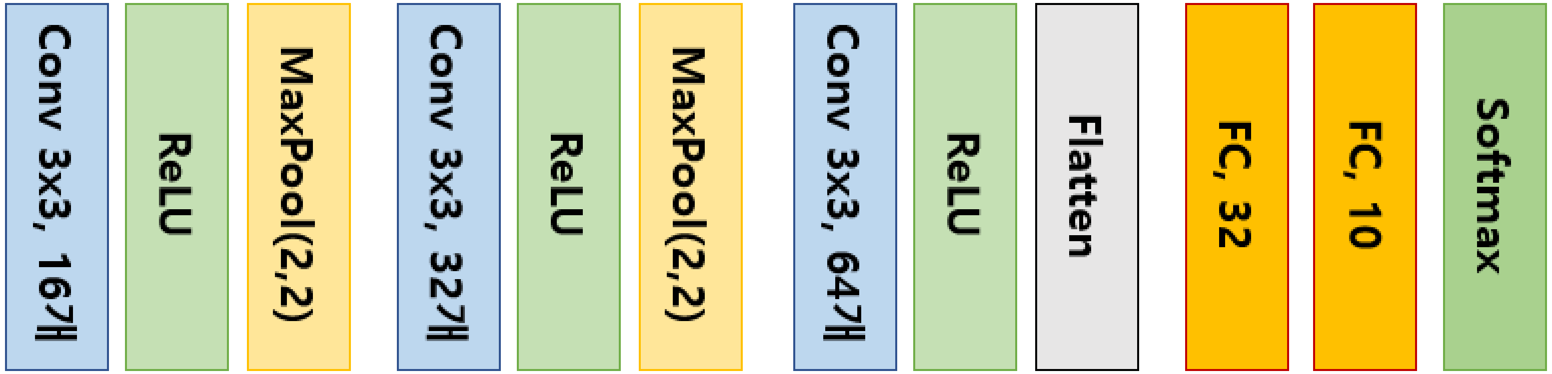
$$Loss = (y - z)^2 = \sum_{i=0}^{n}(y_i - z_i)^2$$

$$\frac{\partial Loss}{\partial w} = 2x(wx + b - y)$$

```python
import tensorflow as tf

w = tf.Variable(1.0)
b = tf.Variable(0.5)
print(w.trainable, b.trainable)

x = tf.convert_to_tensor([1.4])
y = tf.convert_to_tensor([2.1])

with tf.GradientTape() as tape:
    z = tf.add(tf.multiply(w, x), b)
    loss = tf.reduce_sum(tf.square(y - z))

dloss_dw = tape.gradient(loss, w)

tf.print('dL/dw : ', dloss_dw)
```

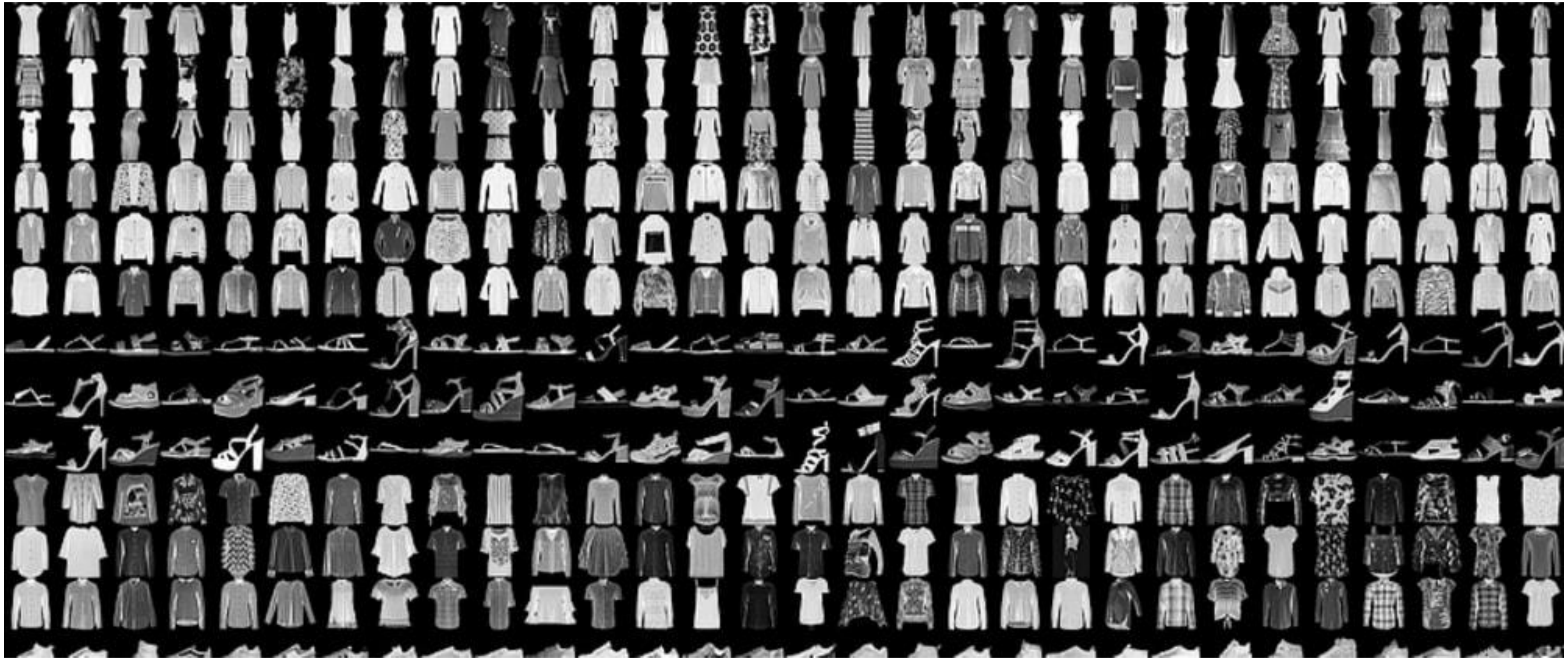
Model Subclassing 실습

model_subclassing.ipynb



Model Subclassing 실습

- Fashion MNIST 데이터셋은 운동화, 셔츠, 샌들과 같은 작은 이미지들의 모음입니다.
- 기본 MNIST 데이터셋과 같이 열 가지로 분류될 수 있는 28×28 픽셀의 이미지 70,000개로 이루어져 있습니다.

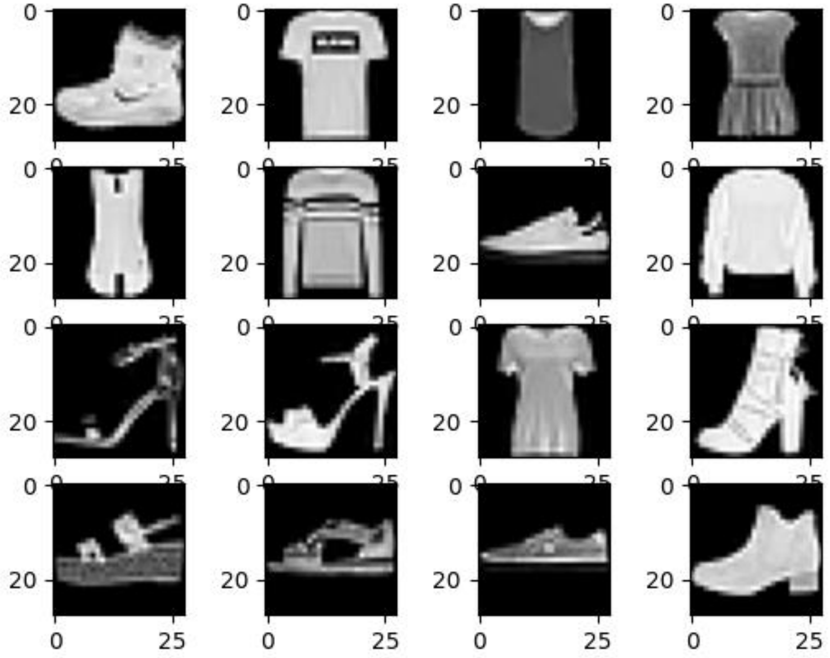


Model Subclassing 실습

```python
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# load fashion_mnist data
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()


# adjusting to 0 ~ 1.0
x_train = x_train / 255.0
x_test = x_test / 255.0

print(x_train.shape, x_test.shape)
```

Model Subclassing 실습

```python
# reshaping
x_train = x_train.reshape(-1,28,28,1)
x_test = x_test.reshape(-1,28,28,1)

# plotting
plt.figure()
for c in range(16):
    plt.subplot(4,4,c+1)
    plt.imshow(x_train[c].reshape(28,28), cmap='gray')
plt.show()
```



Model Subclassing 실습

```
class MyModel(tf.keras.Model):
    def __init__(self):
        super(MyModel, self).__init__()
        self.conv1 = Conv2D(kernel_size=(3,3), filters=16, activation='relu')
        self.conv2 = Conv2D(kernel_size=(3,3), filters=32, activation='relu')
        self.conv3 = Conv2D(kernel_size=(3,3), filters=64, activation='relu')
        self.pool = MaxPooling2D((2, 2))
        self.flatten = Flatten()
        self.d1 = Dense(32, activation='relu')
        self.d2 = Dense(10, activation='softmax')

    def call(self, x):
        x = self.conv1(x)
        x = self.pool(x)
        x = self.conv2(x)
        x = self.pool(x)
        x = self.conv3(x)
        x = self.flatten(x)
        x = self.d1(x)
        return self.d2(x)
```

Model Subclassing 실습

```
model = MyModel()

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


history = model.fit(x_train, y_train,
    epochs=10, validation_split=0.25)

plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r-', label='val_loss')
plt.xlabel('epoch')
plt.legend()
```

Gradient 및 자동 미분 실습

https://www.tensorflow.org/guide/autodiff?hl=ko

GradientTape 실습

model_subclassing_gt.ipynb

GradientTape 실습

Loss Function 정의

```
[12]  loss_function = SparseCategoricalCrossentropy()
```

Optimizer 정의

```
[13]  optimizer = Adam()
```

Metric 정의

```
[14]  train_loss = Mean()
      train_acc = SparseCategoricalAccuracy()
      test_loss = Mean()
      test_acc = SparseCategoricalAccuracy()
```

GradientTape 실습

Train step 함수

```
[15] @tf.function
     def train_step(images, labels):
         with tf.GradientTape() as tape:
             # 1. 예측(Prediction)
             predictions = model(images)
             # 2. Loss 계산
             loss = loss_function(labels, predictions)

         # 3. Gradients 계산
         gradients = tape.gradient(loss, model.trainable_variables)

         # 4. 오차역전파(Backpropagation) - weight 업데이트
         optimizer.apply_gradients(zip(gradients, model.trainable_variables))

         # loss와 accuracy를 업데이트 합니다.
         train_loss(loss)
         train_acc(labels, predictions)
```

GradientTape 실습

Test step 함수

```
[16] @tf.function
     def test_step(images, labels):
         # 1. 예측(Prediction)
         predictions = model(images)
         # 2. Loss 계산
         loss = loss_function(labels, predictions)

         # Test셋에 대해서는 gradient를 계산 및 backpropagation 하지 않습니다.

         # loss와 accuracy를 업데이트
         test_loss(loss)
         test_acc(labels, predictions)
```

GradientTape 실습

모델 훈련

```
[17] EPOCHS = 10

     for epoch in range(EPOCHS):
         for train_images, train_labels in train_ds:
             train_step(train_images, train_labels)

         for test_images, test_labels in test_ds:
             test_step(test_images, test_labels)

         template = 'Epoch : {}, Train Loss : {:.5f}, 정확도: {:.2f}%, 테스트 손실: {:.5f}, 테스트
         print (template.format(epoch+1,
                                train_loss.result(),
                                train_acc.result()*100,
                                test_loss.result(),
                                test_acc.result()*100))
```

```
Epoch : 1, Train Loss : 0.57843, 정확도: 78.79%, 테스트 손실: 0.42673, 테스트 정확도: 84.77%
Epoch : 2, Train Loss : 0.46914, 정확도: 82.85%, 테스트 손실: 0.39928, 테스트 정확도: 85.69%
```

Thank you

kgpark88@gmail.com