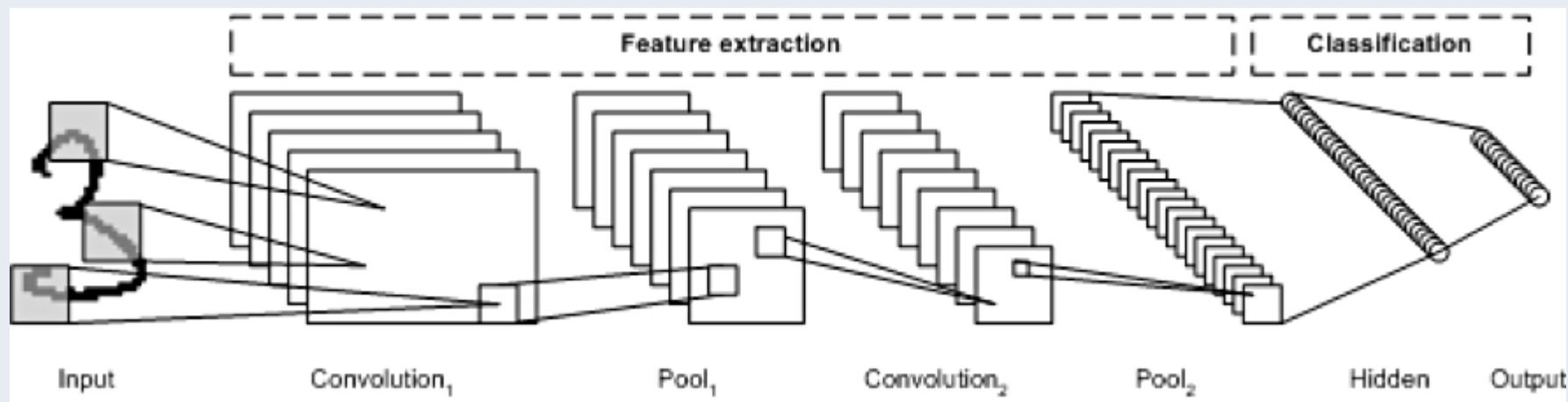
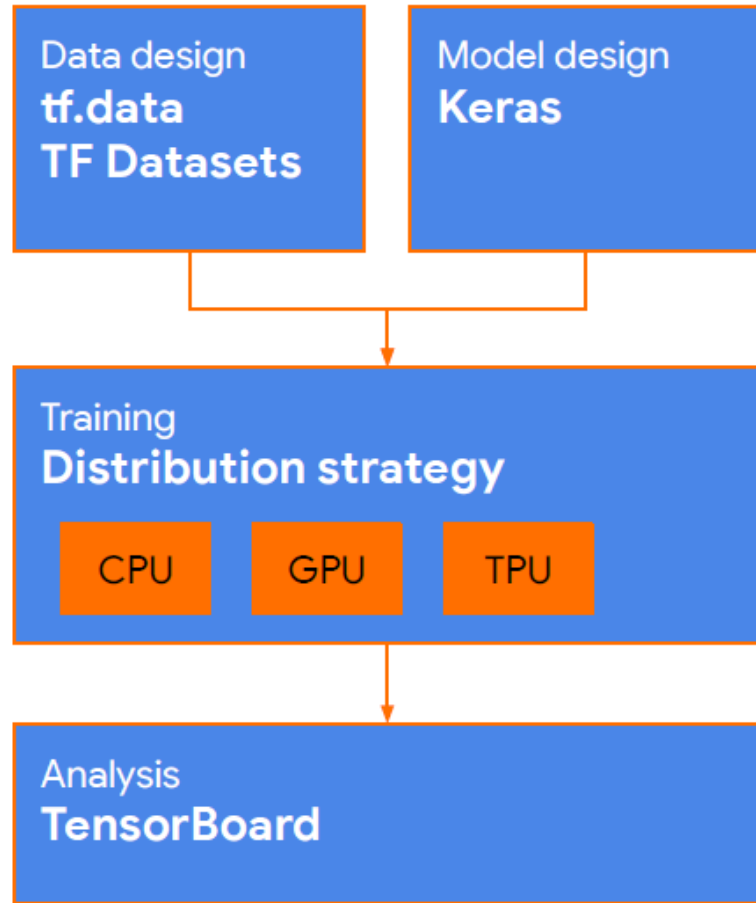


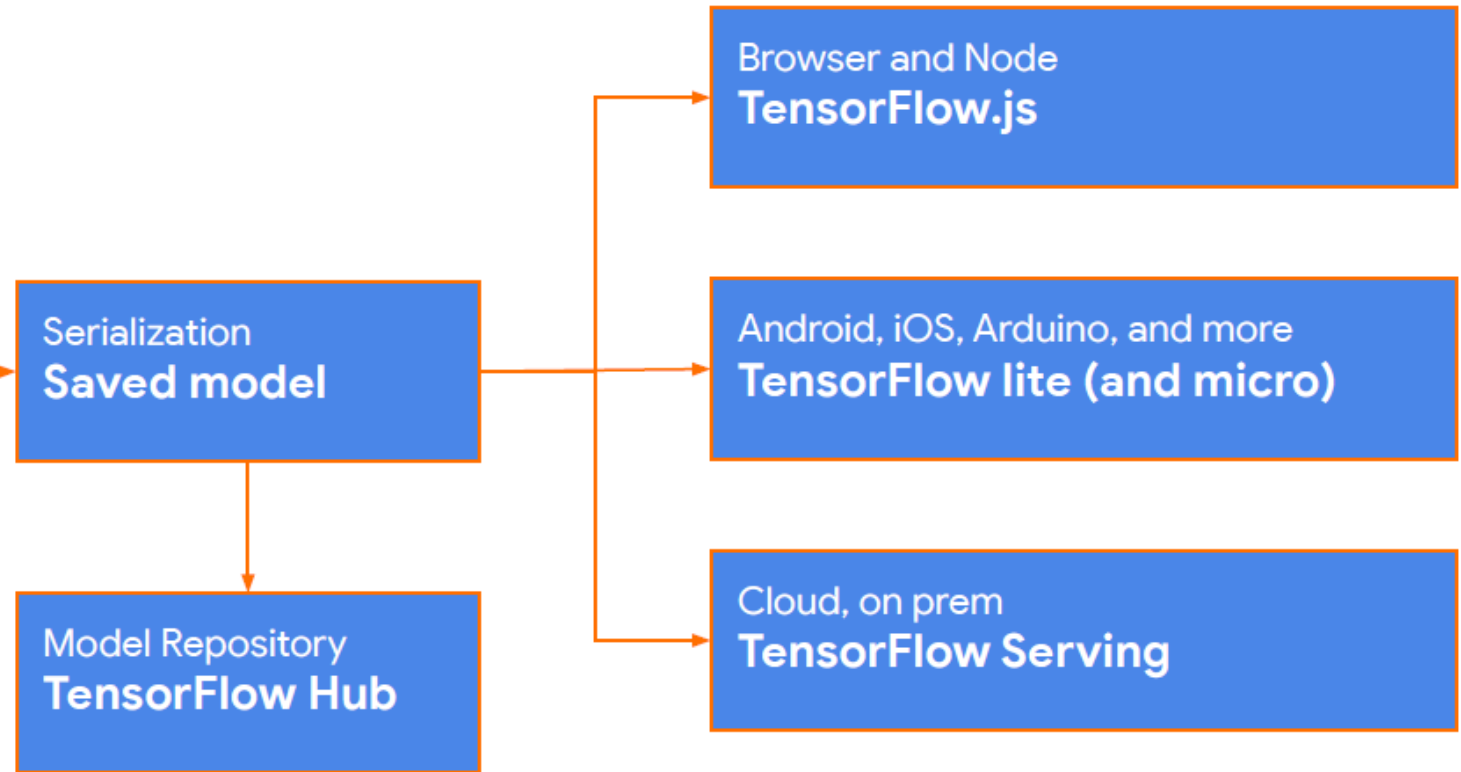
# 모델 서브클래싱



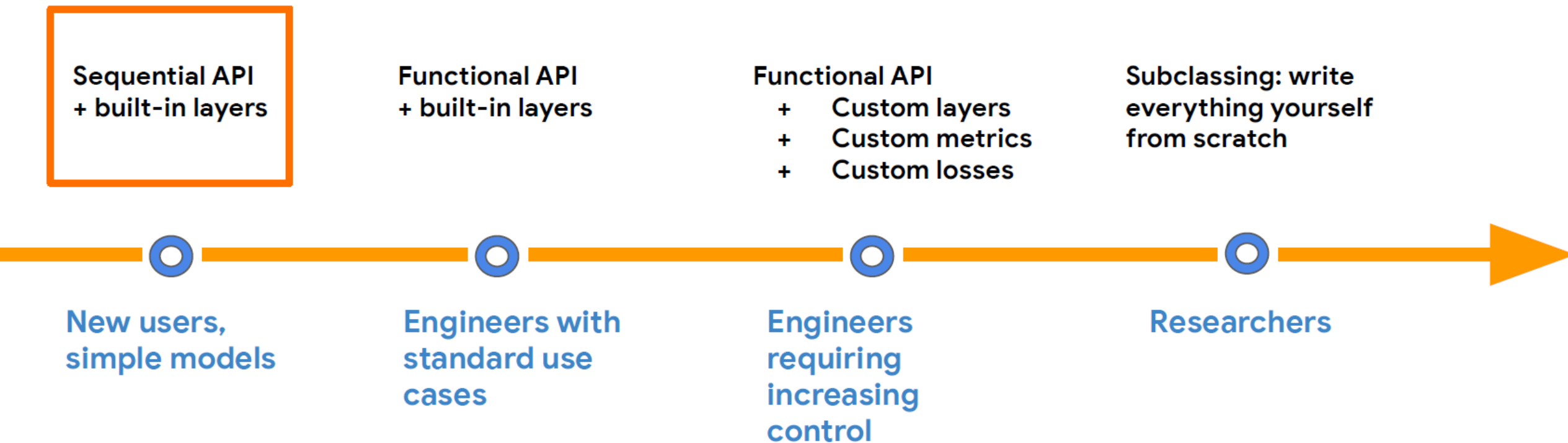
## Training



## Deployment

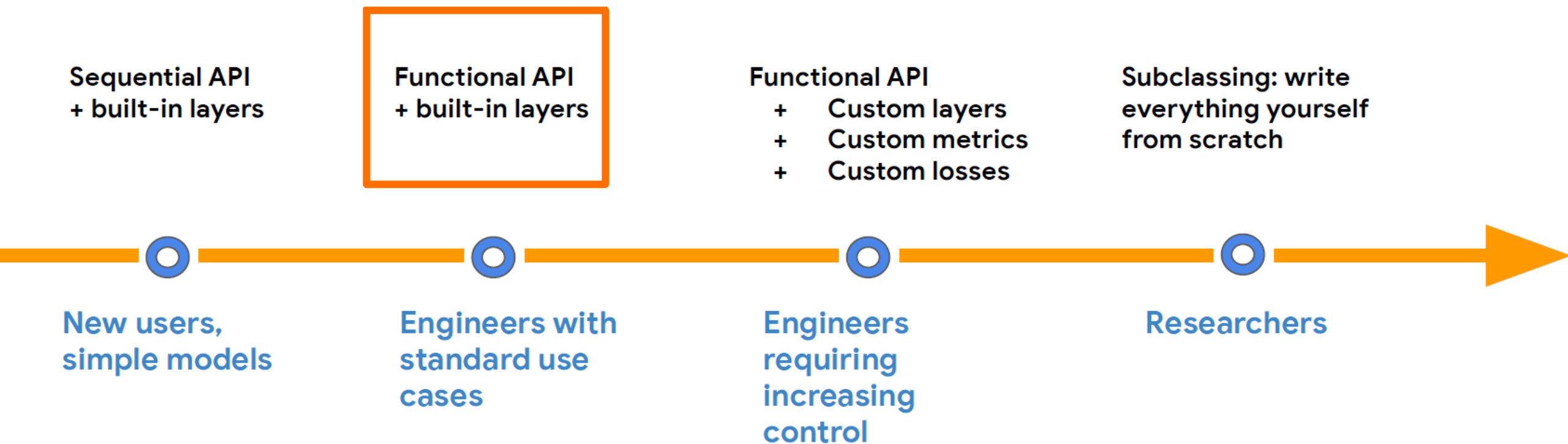


# Model building



```
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'))
```

# Model building



# Model building

## Visual Question Answering



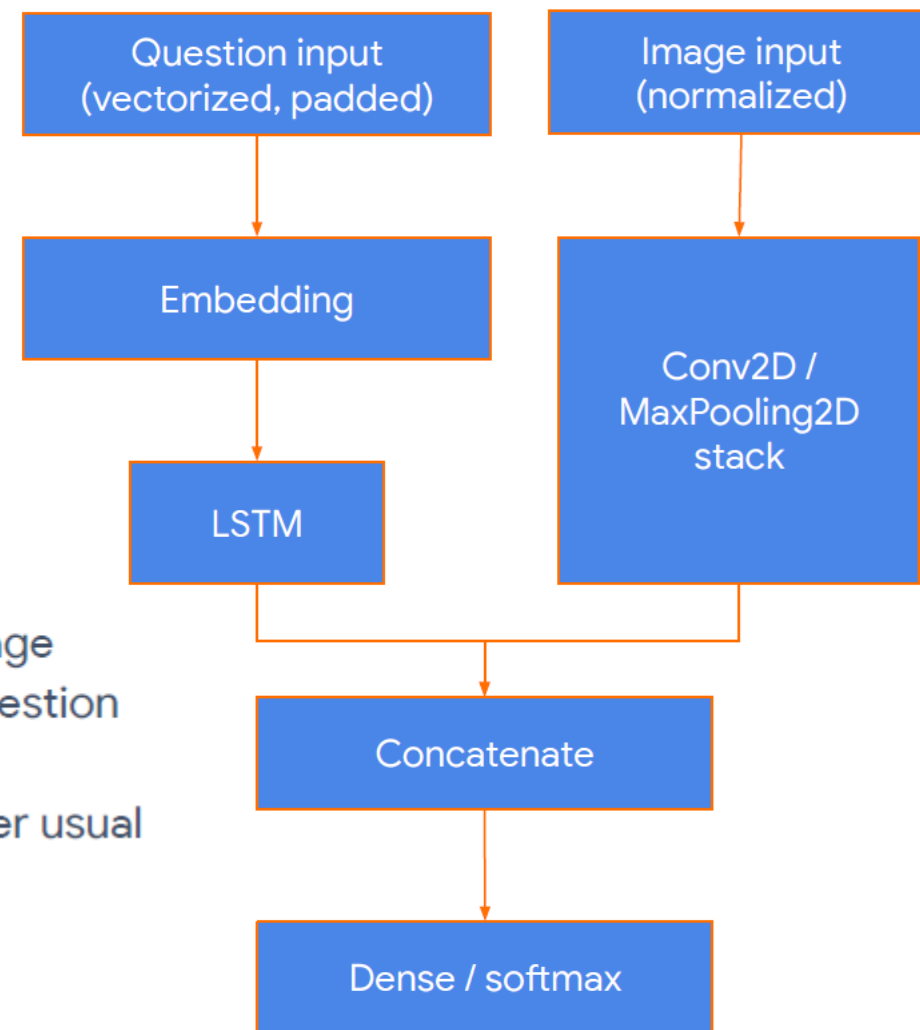
**Question:** What color is the dog on the right?

**Answer:** Golden

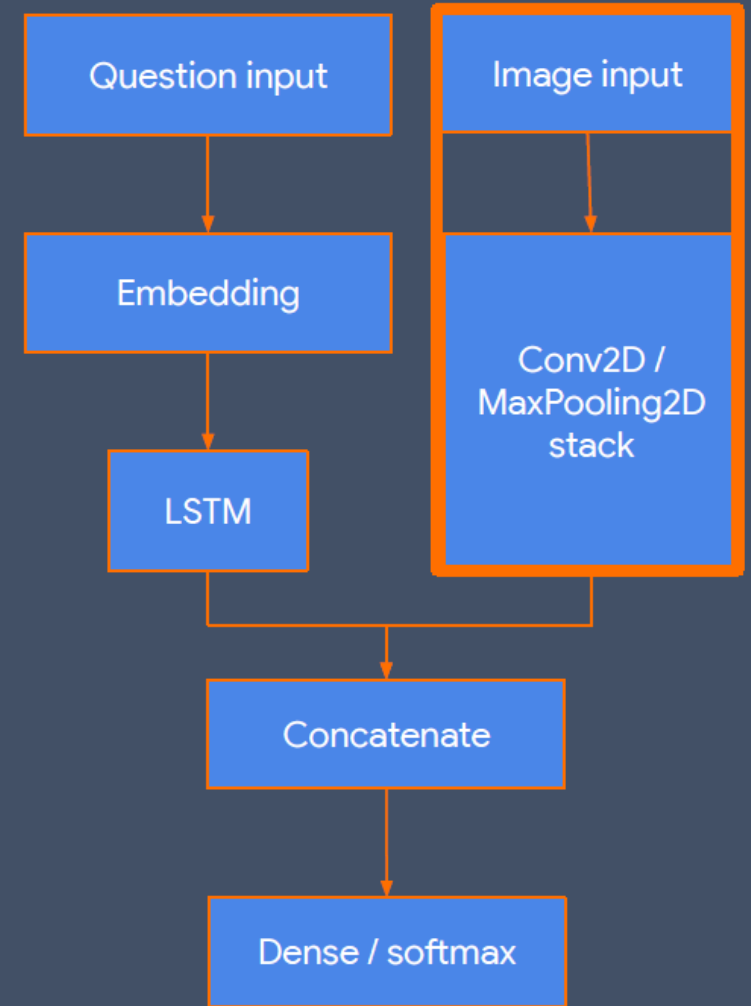
## Workflow

A multi-input model

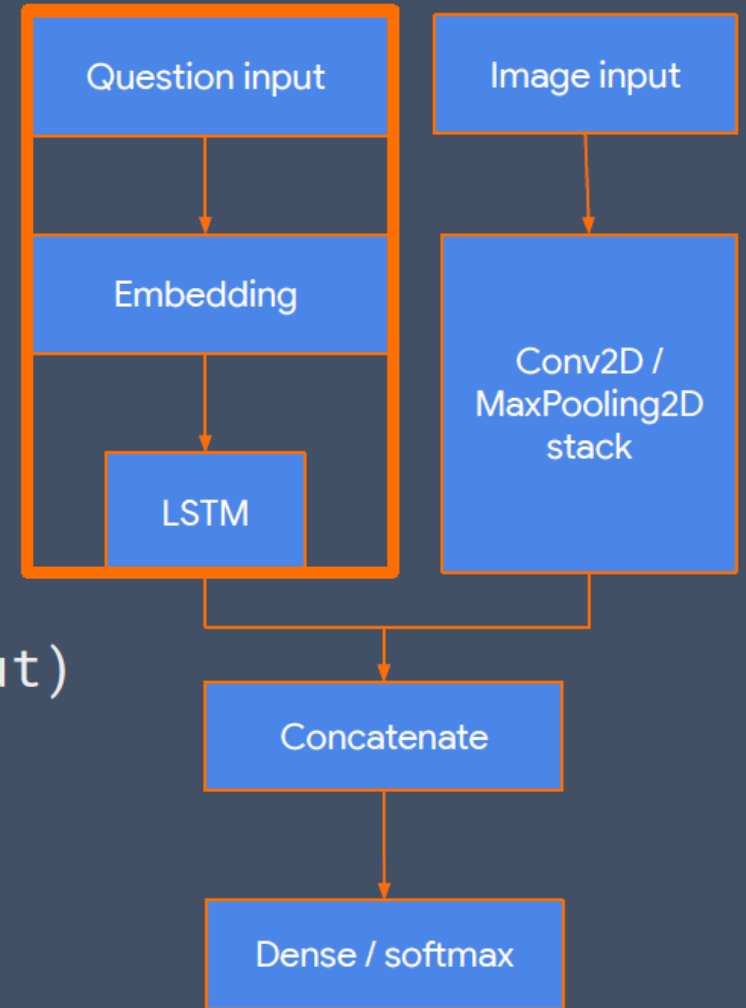
1. Use a CNN to embed the image
2. Use a LSTM to embed the question
3. **Concatenate**
4. Classify with Dense layers, per usual



```
# 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)
```

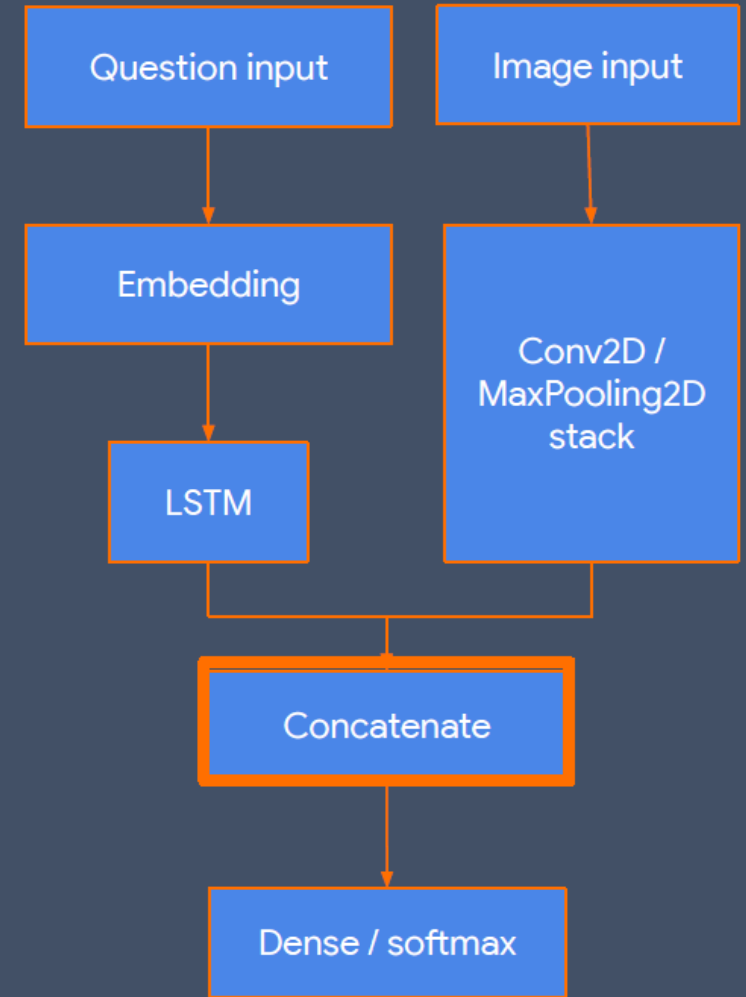


```
# A language model.  
# Encode the question into a vector.  
question_input = Input(shape=(100,),  
                        dtype='int32',  
                        name="Question")  
  
embedded = Embedding(input_dim=10000,  
                    output_dim=256,  
                    input_length=100)(question_input)  
  
encoded_question = LSTM(256)(embedded_question)
```



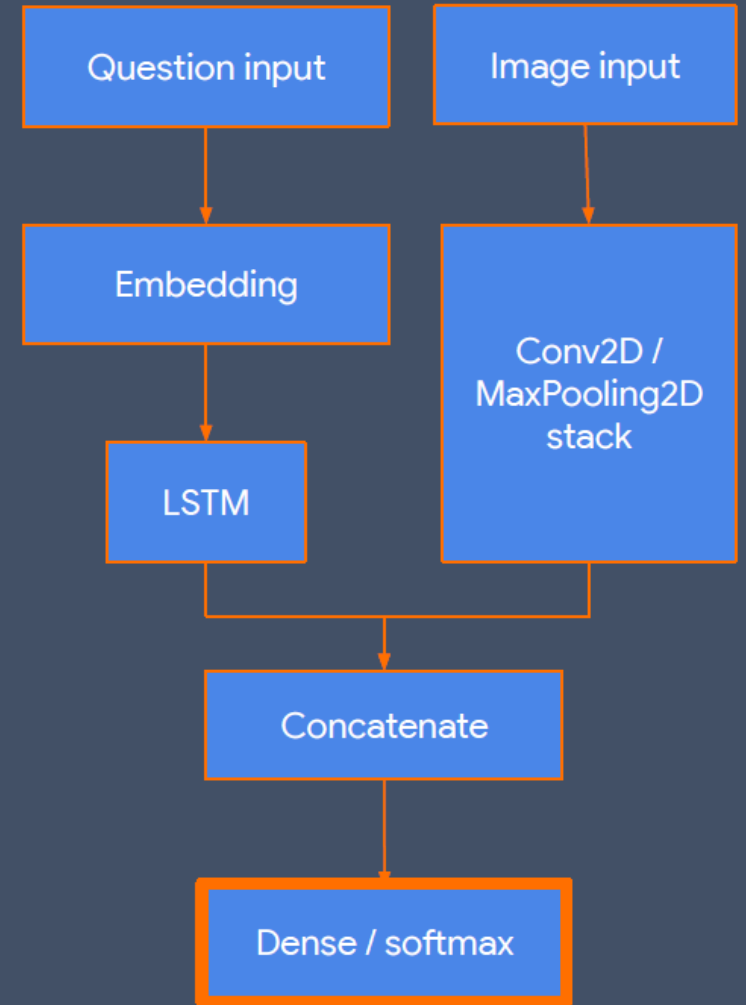


```
# Concatenate the encoded image and question  
merged = layers.concatenate([encoded_image,  
                             encoded_question])
```

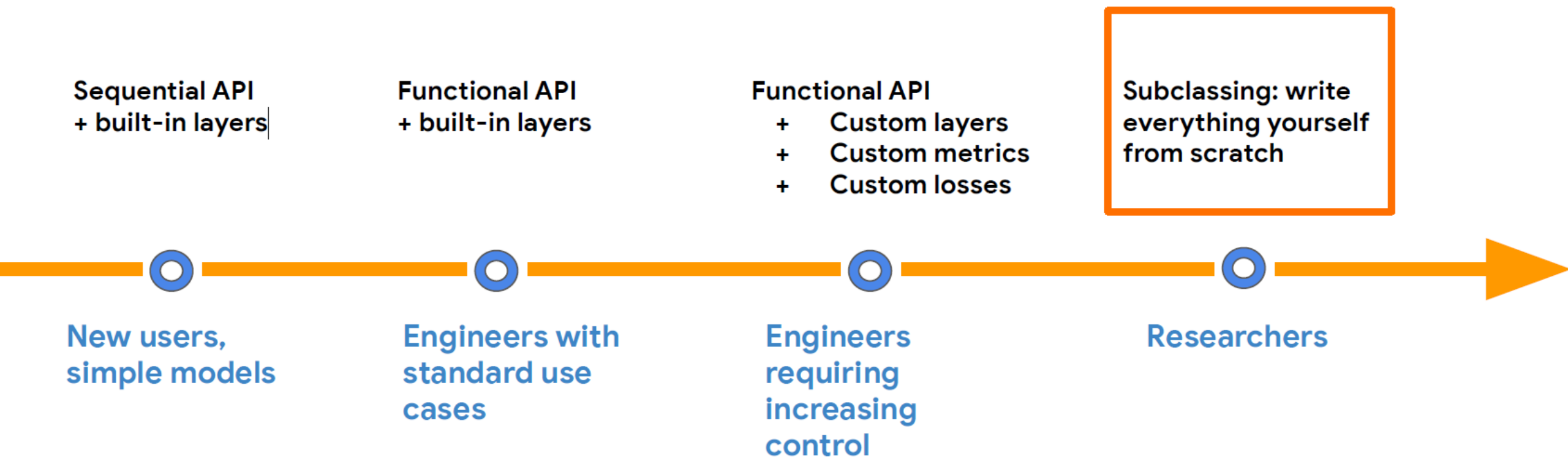


```
# Train a classifier on top.  
output = Dense(1000,  
               activation='softmax')(merged)
```

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



# Model building

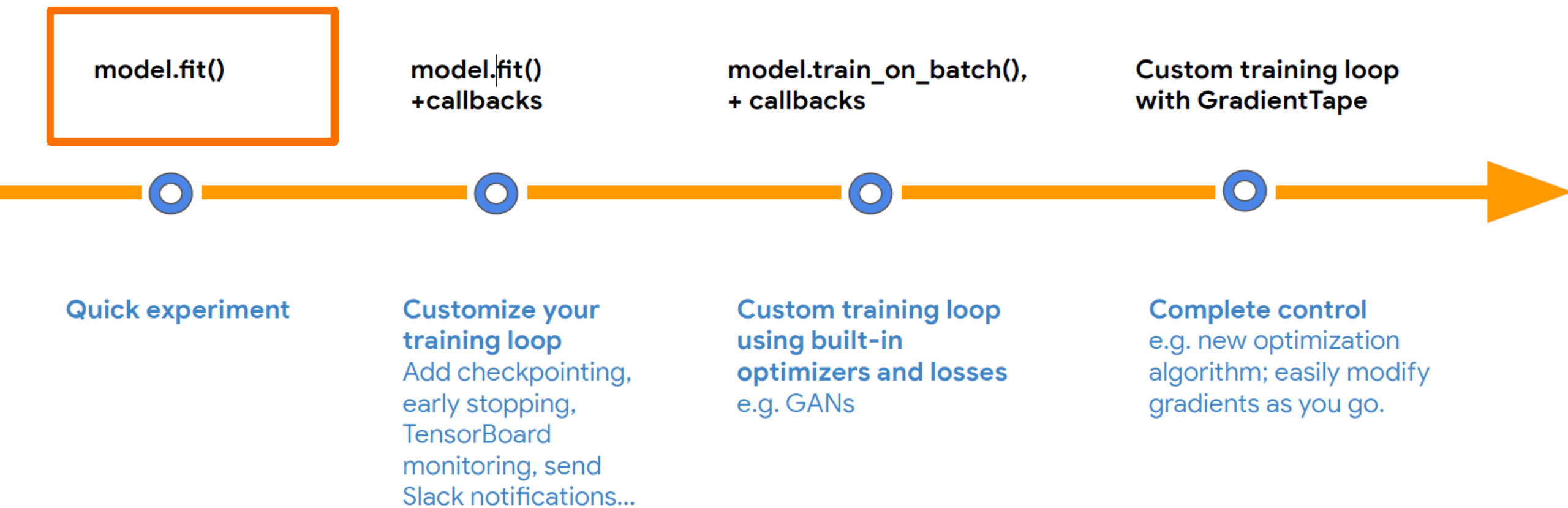


```
class MyModel(tf.keras.Model):  
    def __init__(self, num_classes=10):  
        super(MyModel, self).__init__(name='my_model')  
        self.dense_1 = layers.Dense(32, activation='relu')  
        self.dense_2 = layers.Dense(num_classes, activation='softmax')  
  
    def call(self, inputs):  
        # Define your forward pass here  
        x = self.dense_1(inputs)  
        return self.dense_2(x)
```

```
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

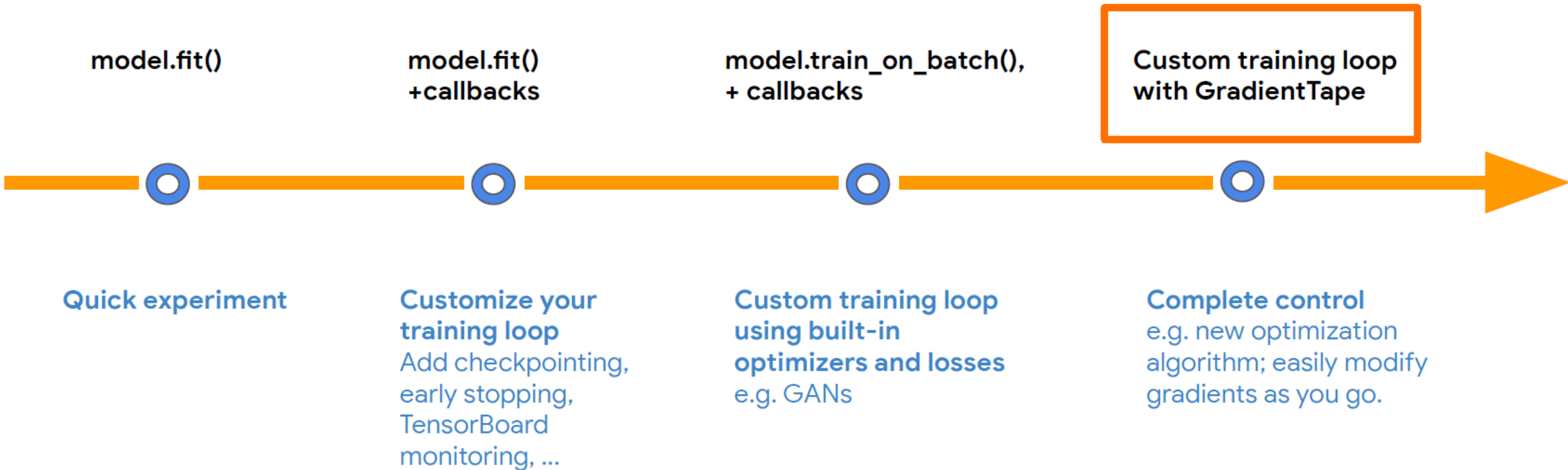


```
model.compile(optimizer=Adam(),  
              loss=BinaryCrossentropy(),  
              metrics=[AUC(), Precision(), Recall()])
```

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

**...or write your own callbacks!**

# Model training





# 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
```

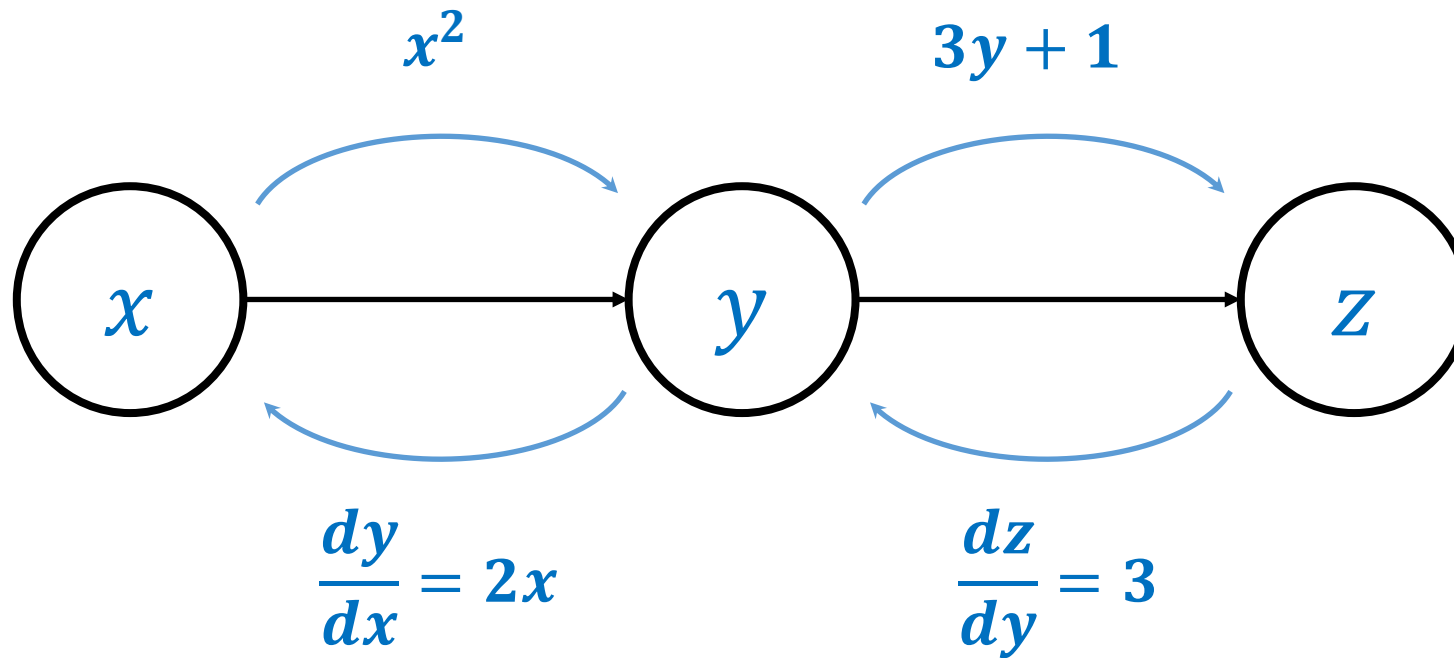
# Variables are automatically tracked

```
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)
```

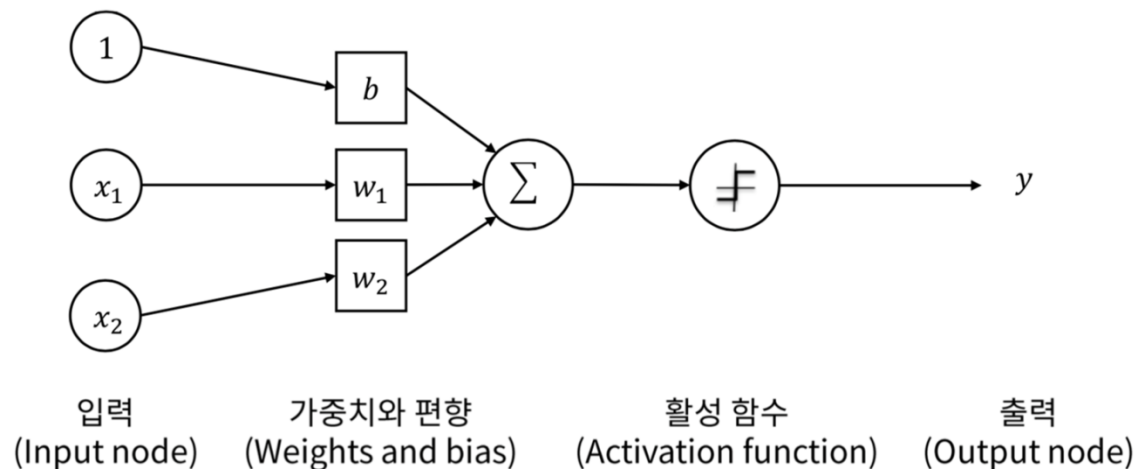
As we saw previously, you can get the gradients for `model.trainable_variables` as well in one go.

# 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.



$$z = wx + b$$

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

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

```
1 import tensorflow as tf
2
3 w = tf.Variable(1.0)
4 b = tf.Variable(0.5)
5 print(w.trainable, b.trainable)
6
7 x = tf.convert_to_tensor([1.4])
8 y = tf.convert_to_tensor([2.1])
9
10 with tf.GradientTape() as tape:
11     z = tf.add(tf.multiply(w, x), b)
12     loss = tf.reduce_sum(tf.square(y - z))
13
14 dloss_dw = tape.gradient(loss, w)
15
16 tf.print('dL/dw : ', dloss_dw)
```

# Model Subclassing 실습

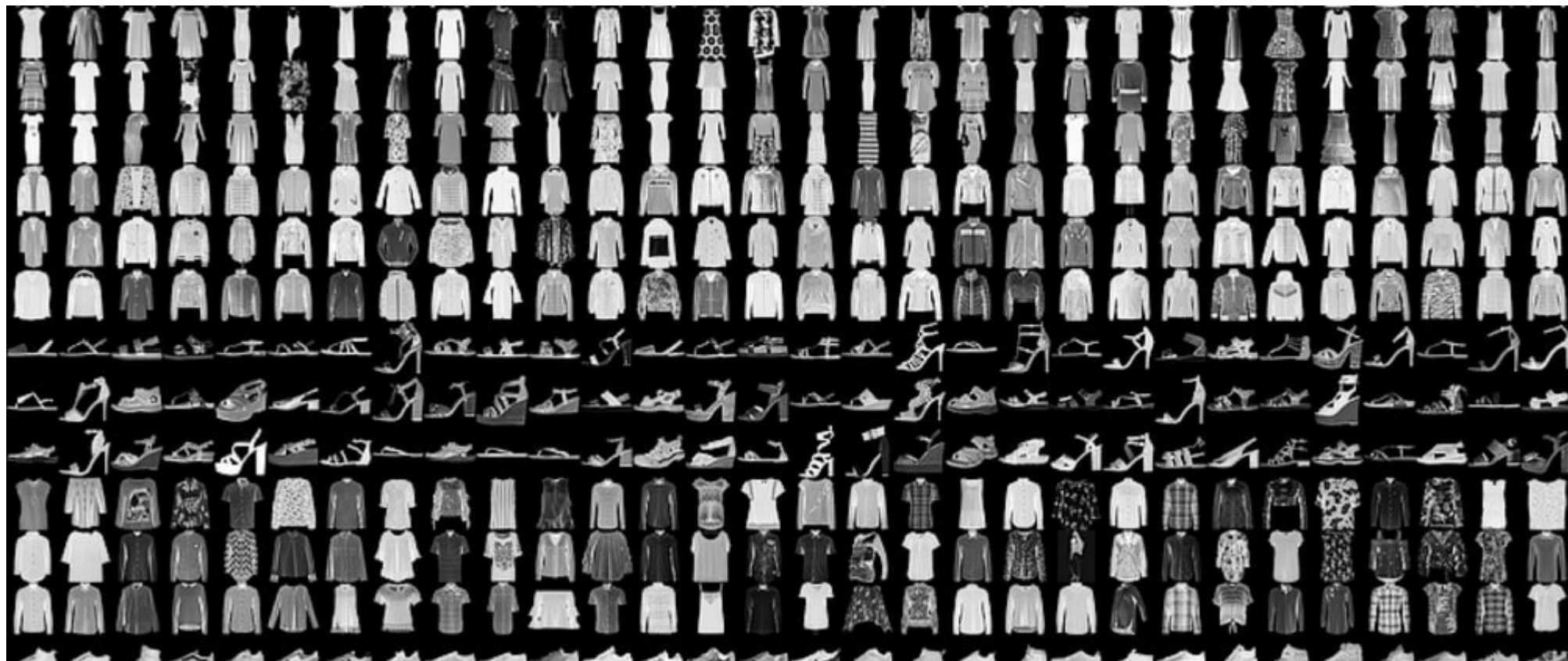


model\_subclassing.ipynb



# 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.



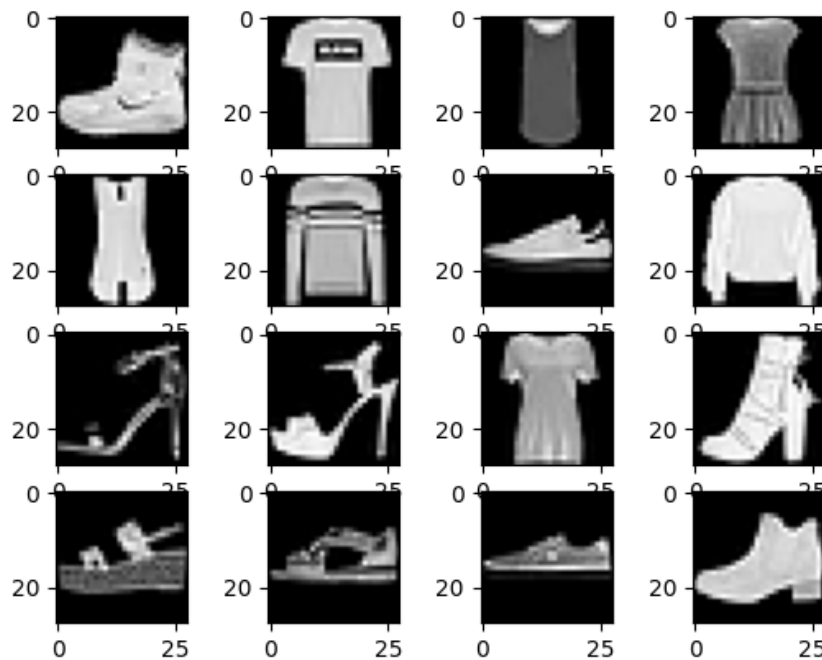
# Model Subclassing 실습

```
1 import matplotlib.pyplot as plt
2
3 import tensorflow as tf
4 from tensorflow.keras.datasets import fashion_mnist
5 from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
6
7 # load fashion_mnist data
8 (x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
9
10
11 # adjusting to 0 ~ 1.0
12 x_train = x_train / 255.0
13 x_test = x_test / 255.0
14
15 print(x_train.shape, x_test.shape)
```



# Model Subclassing 실습

```
17 # reshaping
18 x_train = x_train.reshape(-1,28,28,1)
19 x_test = x_test.reshape(-1,28,28,1)
20
21 # plotting
22 plt.figure()
23 for c in range(16):
24     plt.subplot(4,4,c+1)
25     plt.imshow(x_train[c].reshape(28,28), cmap='gray')
26 plt.show()
```



# Model Subclassing 실습

```
28 class MyModel(tf.keras.Model):
29     def __init__(self):
30         super(MyModel, self).__init__()
31         self.conv1 = Conv2D(kernel_size=(3,3), filters=16, activation='relu')
32         self.conv2 = Conv2D(kernel_size=(3,3), filters=32, activation='relu')
33         self.conv3 = Conv2D(kernel_size=(3,3), filters=64, activation='relu')
34         self.pool = MaxPooling2D((2, 2))
35         self.flatten = Flatten()
36         self.d1 = Dense(32, activation='relu')
37         self.d2 = Dense(10, activation='softmax')
38
39     def call(self, x):
40         x = self.conv1(x)
41         x = self.pool(x)
42         x = self.conv2(x)
43         x = self.pool(x)
44         x = self.conv3(x)
45         x = self.flatten(x)
46         x = self.d1(x)
47         return self.d2(x)
```

# Model Subclassing 실습

```
49 model = MyModel()
50
51 # compile and train
52 model.compile(optimizer='adam',
53               loss='sparse_categorical_crossentropy',
54               metrics=['accuracy'])
55
56
57 history = model.fit(x_train, y_train,
58                    epochs=10, validation_split=0.25)
59
60 plt.figure(figsize=(10,4))
61 plt.subplot(1,2,1)
62 plt.plot(history.history['loss'], 'b-', label='loss')
63 plt.plot(history.history['val_loss'], 'r-', label='val_loss')
64 plt.xlabel('epoch')
65 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](mailto:kgpark88@gmail.com)