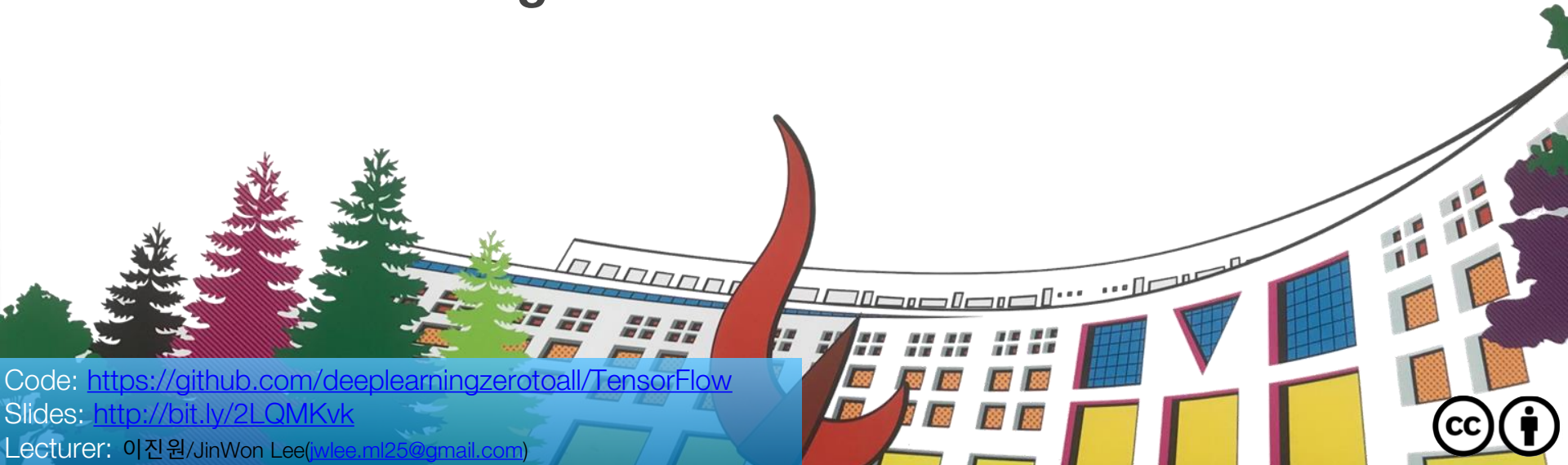
ML/DL for Everyone Season2

with TensorFlow

Lab 11-2 CNN with MNIST Dataset using tf.keras Functional APIs

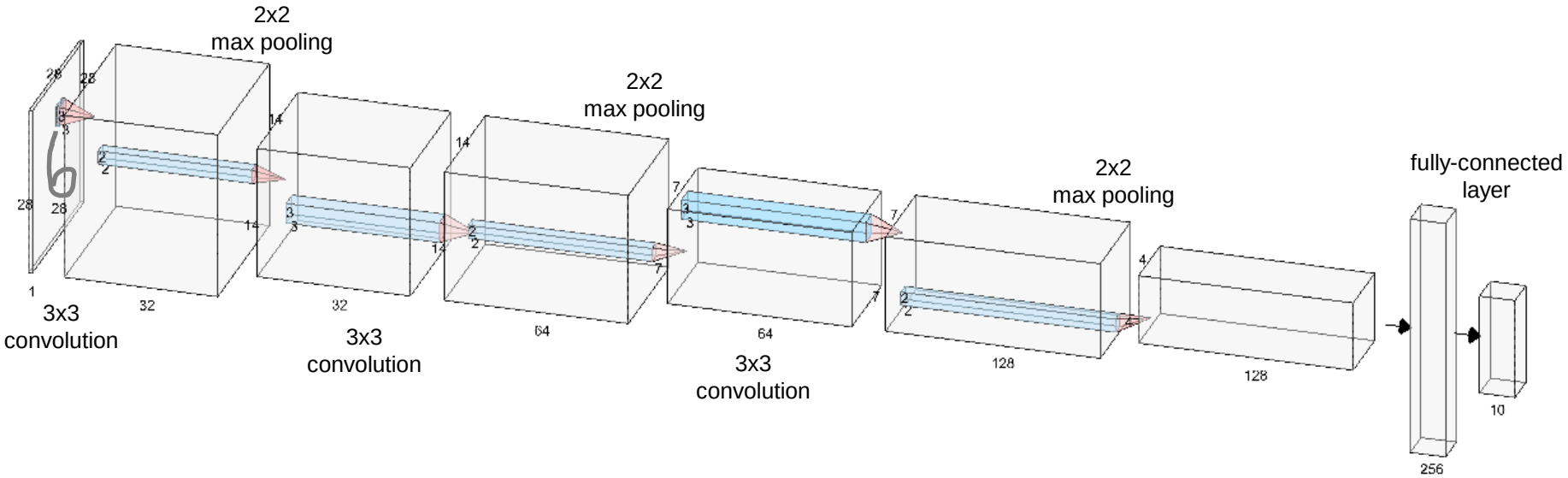


Code: https://github.com/deeplearningzerotoall/TensorFlow
Slides: http://bit.ly/2LQMKvk
Lecturer: 이진원/JinWon Lee(jwlee.ml25@gmail.com)

NN Implementation Flow in TensorFlow

1. Set hyper parameters – learning rate, training epochs, batch size, etc.
2. Make a data pipelining – use tf.data
3. Build a neural network model – use tf.keras functional APIs
4. Define a loss function – cross entropy
5. Calculate a gradient – use tf.GradientTape
6. Select an optimizer – Adam optimizer
7. Define a metric for model's performance – accuracy
8. (optional) Make a checkpoint for saving
9. Train and Validate a neural network model

CNN with MNIST Data



0. Import Libraries

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt
import os
```

1. Set Hyper Parameters

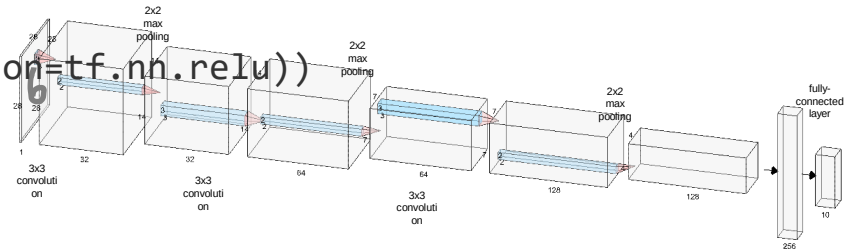
```
learning_rate = 0.001
training_epochs = 15
batch_size = 100


cur_dir = os.getcwd()
ckpt_dir_name = 'checkpoints'
model_dir_name = 'minst_cnn_func'

checkpoint_dir = os.path.join(cur_dir, ckpt_dir_name, model_dir_name)
os.makedirs(checkpoint_dir, exist_ok=True)

checkpoint_prefix = os.path.join(checkpoint_dir, model_dir_name)
```

2. Make a Data Pipelining

```
mnist = keras.datasets.mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.astype(np.float32) / 255.
test_images = test_images.astype(np.float32) / 255.
train_images = np.expand_dims(train_images, axis=-1)
test_images = np.expand_dims(test_images, axis=-1)

train_labels = to_categorical(train_labels, 10)
test_labels = to_categorical(test_labels, 10)

train_dataset = tf.data.Dataset.from_tensor_slices((train_images,
        train_labels)).shuffle(buffer_size=100000).batch(batch_size)
test_dataset = tf.data.Dataset.from_tensor_slices((test_images,
        test_labels)).batch(batch_size)
```

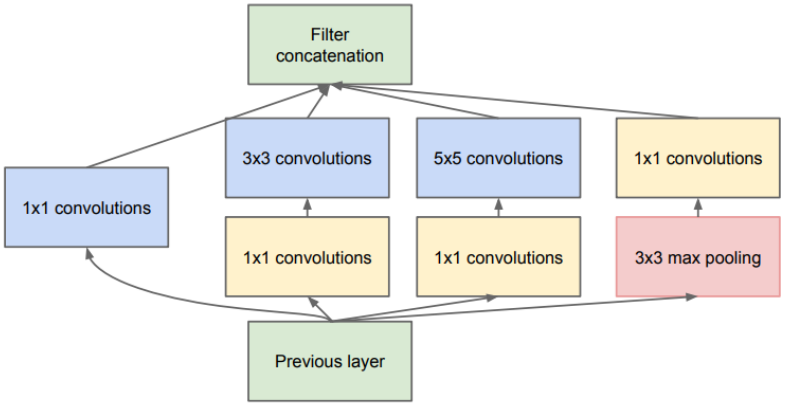
3. Build a Neural Network Model – Sequential API

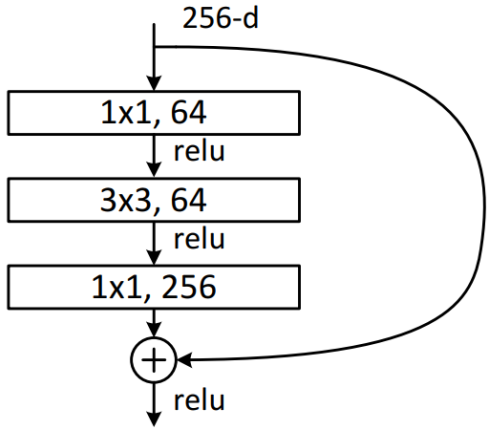
```
def create_model():
    model = keras.Sequential()
    model.add(keras.layers.Conv2D(filters=32, kernel_size=3, activation=tf.nn.relu,
          padding='SAME', input_shape=(28, 28, 1)))
    model.add(keras.layers.MaxPool2D(padding='SAME'))
    model.add(keras.layers.Conv2D(filters=64, kernel_size=3, activation=tf.nn.relu,
          padding='SAME'))
    model.add(keras.layers.MaxPool2D(padding='SAME'))
    model.add(keras.layers.Conv2D(filters=128, kernel_size=3, activation=tf.nn.relu,
          padding='SAME'))
    model.add(keras.layers.MaxPool2D(padding='SAME'))
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(256, activation=tf.nn.relu))
    model.add(keras.layers.Dropout(0.4))
    model.add(keras.layers.Dense(10))
    return model
```



Limitation of Sequential API

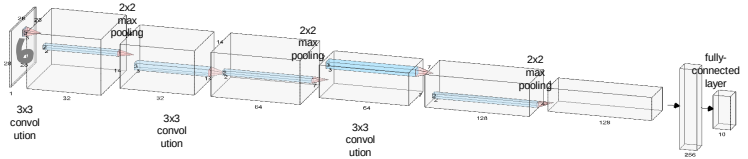
- We can’t make
  - Multi-input models
  - Multi-output models
  - Models with shared layers(the same layer called several times)
  - Models with non-sequential data flow (e.g.. Residual connections)





3. Build a Neural Network Model – Functional API

```
def create_model():
    inputs = keras.Input(shape=(28, 28, 1))
    conv1 = keras.layers.Conv2D(filters=32, kernel_size=3, padding='SAME',
           activation=tf.nn.relu)(inputs)
    pool1 = keras.layers.MaxPool2D(padding='SAME')(conv1)
    conv2 = keras.layers.Conv2D(filters=64, kernel_size=3, padding='SAME',
           activation=tf.nn.relu)(pool1)
    pool2 = keras.layers.MaxPool2D(padding='SAME')(conv2)
    conv3 = keras.layers.Conv2D(filters=128, kernel_size=3, padding='SAME',
           activation=tf.nn.relu)(pool2)
    pool3 = keras.layers.MaxPool2D(padding='SAME')(conv3)
    pool3_flat = keras.layers.Flatten()(pool3)
    dense4 = keras.layers.Dense(units=256, activation=tf.nn.relu)(pool3_flat)
    drop4 = keras.layers.Dropout(rate=0.4)(dense4)
    logits = keras.layers.Dense(units=10)(drop4)
    return keras.Model(inputs=inputs, outputs=logits)
```

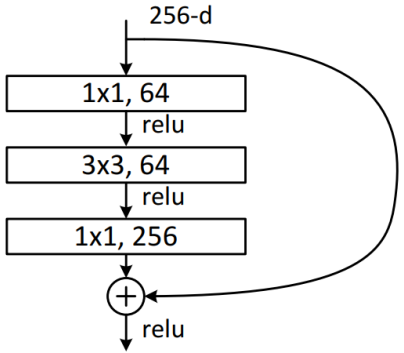


3. Build a Neural Network Model – Functional API

```
model = create_model()
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 28, 28, 1)         0
_________________________________________________________________
conv2d (Conv2D)              (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 128)         73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 256)               524544
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                2570
=================================================================
Total params: 619,786
Trainable params: 619,786
Non-trainable params: 0
_________________________________________________________________
```

Implementation of Residual Block




```
inputs = keras.Input(shape=(28, 28, 256))
conv1 = keras.layers.Conv2D(filters=64, kernel_size=1, padding='SAME',
        activation=keras.layers.ReLU())(inputs)
conv2 = keras.layers.Conv2D(filters=64, kernel_size=3, padding='SAME',
        activation=keras.layers.ReLU())(conv1)
conv3 = keras.layers.Conv2D(filters=256, kernel_size=1, padding='SAME')(conv2)
add3 = keras.layers.add([conv3, inputs])
relu3 = keras.layers.ReLU()(add3)
model = keras.Model(inputs=inputs, outputs=relu3)
```

4. Define a Loss Function
5. Calculate a Gradient

```
def loss_fn(model, images, labels):
   logits = model(images, training=True)
   loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(
           logits=logits, labels=labels))
   return loss


def grad(model, images, labels):
   with tf.GradientTape() as tape:
       loss = loss_fn(model, images, labels)
   return tape.gradient(loss, model.variables)
```

6. Select an Optimizer
7. Define a Metric for Model's Performance
8. Make a Checkpoint for Saving

```
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)

def evaluate(model, images, labels):
    logits = model(images, training=False)
    correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(labels, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    return accuracy

checkpoint = tf.train.Checkpoint(cnn=model)
```

9. Train and Validate a Neural Network Model

```
for epoch in range(training_epochs):
    avg_loss = 0.
    avg_train_acc = 0.
    avg_test_acc = 0.
    train_step = 0
    test_step = 0

    for images, labels in train_dataset:
        grads = grad(model, images, labels)
        optimizer.apply_gradients(zip(grads, model.variables))
        loss = loss_fn(model, images, labels)
        acc = evaluate(model, images, labels)
        avg_loss = avg_loss + loss
        avg_train_acc = avg_train_acc + acc
        train_step += 1
    avg_loss = avg_loss / train_step
    avg_train_acc = avg_train_acc / train_step
```

9. Train and Validate a Neural Network Model

```
for images, labels in test_dataset:
        acc = evaluate(model, images, labels)
        avg_test_acc = avg_test_acc + acc
        test_step += 1
    avg_test_acc = avg_test_acc / test_step

    print('Epoch:', '{}'.format(epoch + 1), 'loss =', '{:.8f}'.format(avg_loss),
          'train accuracy = ', '{:.4f}'.format(avg_train_acc),
          'test accuracy = ', '{:.4f}'.format(avg_test_acc))

    checkpoint.save(file_prefix=checkpoint_prefix)
```

Accuracy : 99.35%

What’s Next?

- CNN with MNIST Dataset using tf.keras.Model subclassing