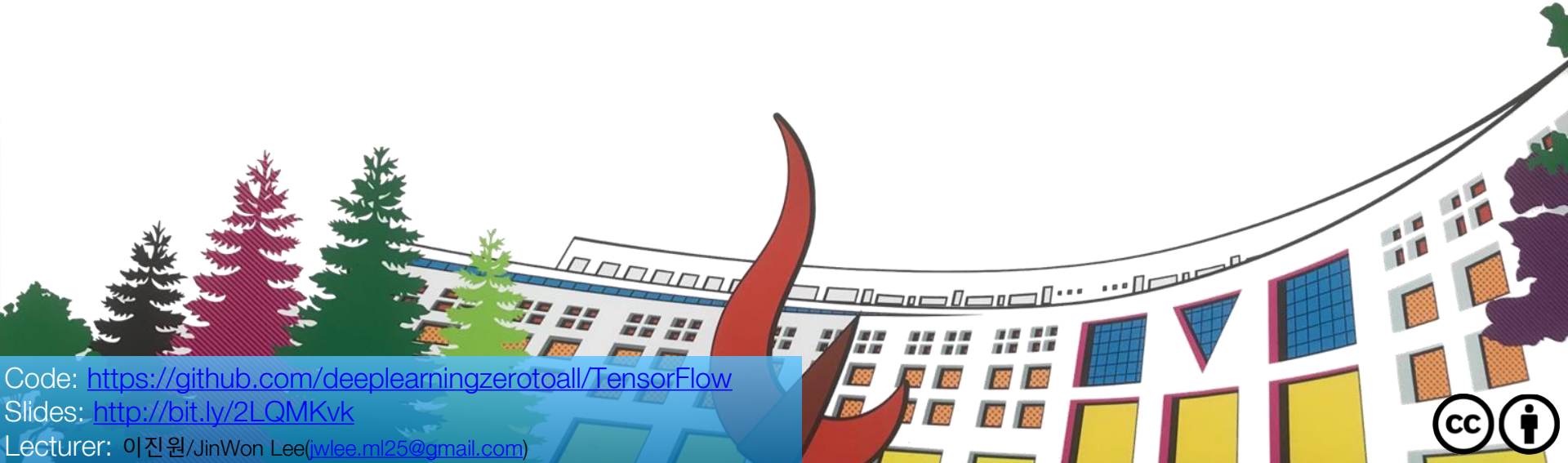
ML/DL for Everyone Season2

with TensorFlow

Lab 11-5 Best CNN with MNIST Dataset

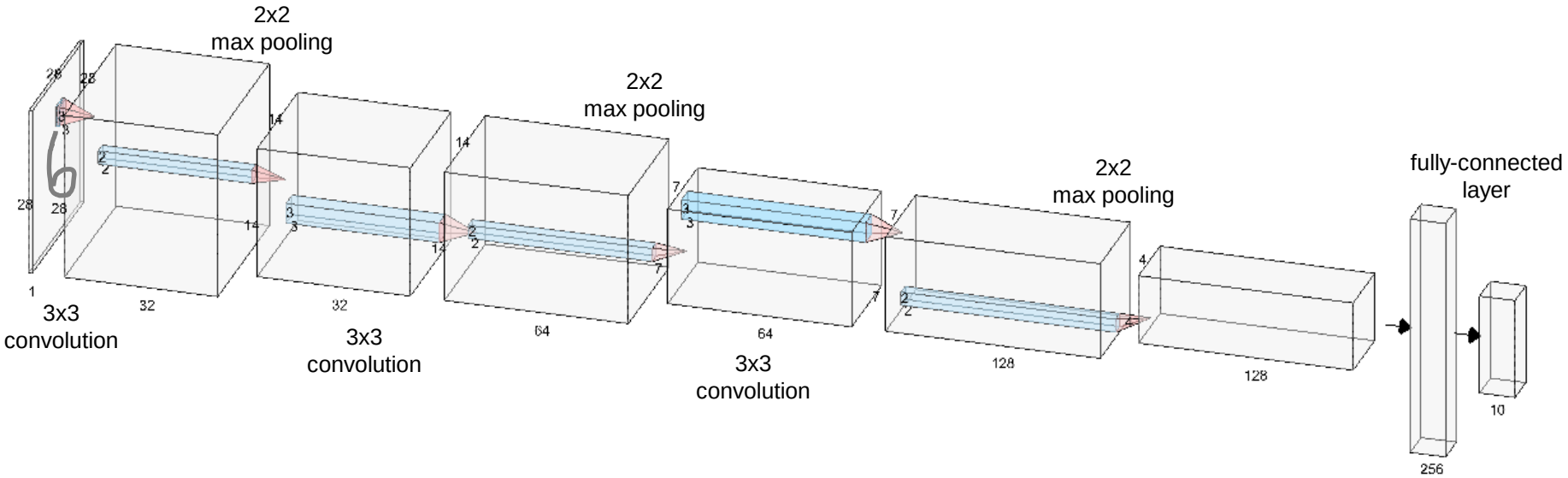


Code: https://github.com/deeplearningzerotoall/TensorFlow
Slides: http://bit.ly/2LQMKvk
Lecturer: 이진원/JinWon Lee(jwlee.ml25@gmail.com)

NN Implementation Flow in TensorFlow

1. Set hyper parameters – learning rate, training epochs, batch size, etc.
2. Data Augmentation – rotate & shift
3. Make a data pipelining – use tf.data
4. Build a neural network model – use tf.keras
5. Define a loss function – cross entropy
6. Calculate a gradient – use tf.GradientTape
7. Select an optimizer – Adam optimizer
8. Define a metric for model's performance – accuracy
9. (optional) Make a checkpoint for saving
10. 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
from scipy import ndimage
```

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_best'

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. Data Augmentation

```
from scipy import ndimage
```

```
def data_augmentation(images, labels):
    aug_images = []; aug_labels = []

    for x, y in zip(images, labels):
        aug_images.append(x)
        aug_labels.append(y)

        bg_value = np.median(x)
        for _ in range(4):
            angle = np.random.randint(-15, 15, 1)
            rot_img = ndimage.rotate(x, angle, reshape=False, cval=bg_value)

            shift = np.random.randint(-2, 2, 2)
            shift_img = ndimage.shift(rot_img, shift, cval=bg_value)

            aug_images.append(shift_img)
            aug_labels.append(y)
    aug_images = np.array(aug_images)
    aug_labels = np.array(aug_labels)
    return aug_images, aug_labels
```

3. Make a Data Pipelining

```
mnist = keras.datasets.mnist

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

train_images, train_labels = data_augmentation(train_images, train_labels)

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=500000).batch(batch_size)
test_dataset = tf.data.Dataset.from_tensor_slices((test_images,
                  test_labels)).batch(batch_size)
```

4. Build a Neural Network Model

```
class ConvBNRelu(tf.keras.Model):
    def __init__(self, filters, kernel_size=3, strides=1, padding='SAME'):
        super(ConvBNRelu, self).__init__()
        self.conv = keras.layers.Conv2D(filters=filters, kernel_size=kernel_size,
                    strides=strides, padding=padding,
                    kernel_initializer='glorot_normal')
        self.batchnorm = tf.keras.layers.BatchNormalization()


    def call(self, inputs, training=False):
        layer = self.conv(inputs)
        layer = self.batchnorm(layer)
        layer = tf.nn.relu(layer)
        return layer
```

4. Build a Neural Network Model

```
class DenseBNRelu(tf.keras.Model):
    def __init__(self, units):
        super(DenseBNRelu, self).__init__()
        self.dense = keras.layers.Dense(units=units,
                    kernel_initializer='glorot_normal')
        self.batchnorm = tf.keras.layers.BatchNormalization()


    def call(self, inputs, training=False):
        layer = self.dense(inputs)
        layer = self.batchnorm(layer)
        layer = tf.nn.relu(layer)
        return layer
```

4. Build a Neural Network Model

```
class MNISTModel(tf.keras.Model):
    def __init__(self):
        super(MNISTModel, self).__init__()
        self.conv1 = ConvBNRelu(filters=32, kernel_size=3, padding='SAME')
        self.pool1 = keras.layers.MaxPool2D(padding='SAME')
        self.conv2 = ConvBNRelu(filters=64, kernel_size=3, padding='SAME')
        self.pool2 = keras.layers.MaxPool2D(padding='SAME')
        self.conv3 = ConvBNRelu(filters=128, kernel_size=3, padding='SAME')
        self.pool3 = keras.layers.MaxPool2D(padding='SAME')
        self.pool3_flat = keras.layers.Flatten()
        self.dense4 = DenseBNRelu(units=256)
        self.drop4 = keras.layers.Dropout(rate=0.4)
        self.dense5 = keras.layers.Dense(units=10,
                                         kernel_initializer='glorot_normal')
```

4. Build a Neural Network Model

```
def call(self, inputs, training=False):
        net = self.conv1(inputs)
        net = self.pool1(net)
        net = self.conv2(net)
        net = self.pool2(net)
        net = self.conv3(net)
        net = self.pool3(net)
        net = self.pool3_flat(net)
        net = self.dense4(net)
        net = self.drop4(net)
        net = self.dense5(net)
        return net

models = []
num_models = 5
for m in range(num_models):
    models.append(MNISTModel())
```

5. Define a Loss Function
6. 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)
```

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

```
global_step = tf.train.get_or_create_global_step()
lr_decay = tf.train.exponential_decay(learning_rate, global_step,
                                      train_images.shape[0]/batch_size*num_models*5,
                                      0.5, staircase=True)
optimizer = tf.train.AdamOptimizer(learning_rate=lr_decay)

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

checkpoints = []
for m in range(num_models):
   checkpoints.append(tf.train.Checkpoint(cnn=models[m]))
```

10. 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:
        for model in models:
            grads = grad(model, images, labels)
            optimizer.apply_gradients(zip(grads, model.variables))
            loss = loss_fn(model, images, labels)
            avg_loss += loss / num_models
        acc = evaluate(models, images, labels)
        avg_train_acc += acc
        train_step += 1
    avg_loss = avg_loss / train_step
    avg_train_acc = avg_train_acc / train_step
```

10. Train and Validate a Neural Network Model

```
for images, labels in test_dataset:
        acc = evaluate(models, images, labels)
        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))

    for idx, checkpoint in enumerate(checkpoints):
        checkpoint.save(file_prefix=checkpoint_prefix+'-{}'.format(idx))
```

Accuracy : 99.68%

What's Next?

- Recurrent Neural Network (RNN)