Tutorial 5 – CNN Introduction and TensorFlow Lite Hands-on (Lab 4-5)

Convolutional Neural Network Introduction

What is a Convolutional Neural Network?

- In computers, images are stored as arrays of numbers. Each image contains features associated with different objects.
- The goal is to make computers recognize objects by extracting and utilizing the underlying features of an image.
- Face recognition and image classification are common applications of CNN.

Example: Image Classification

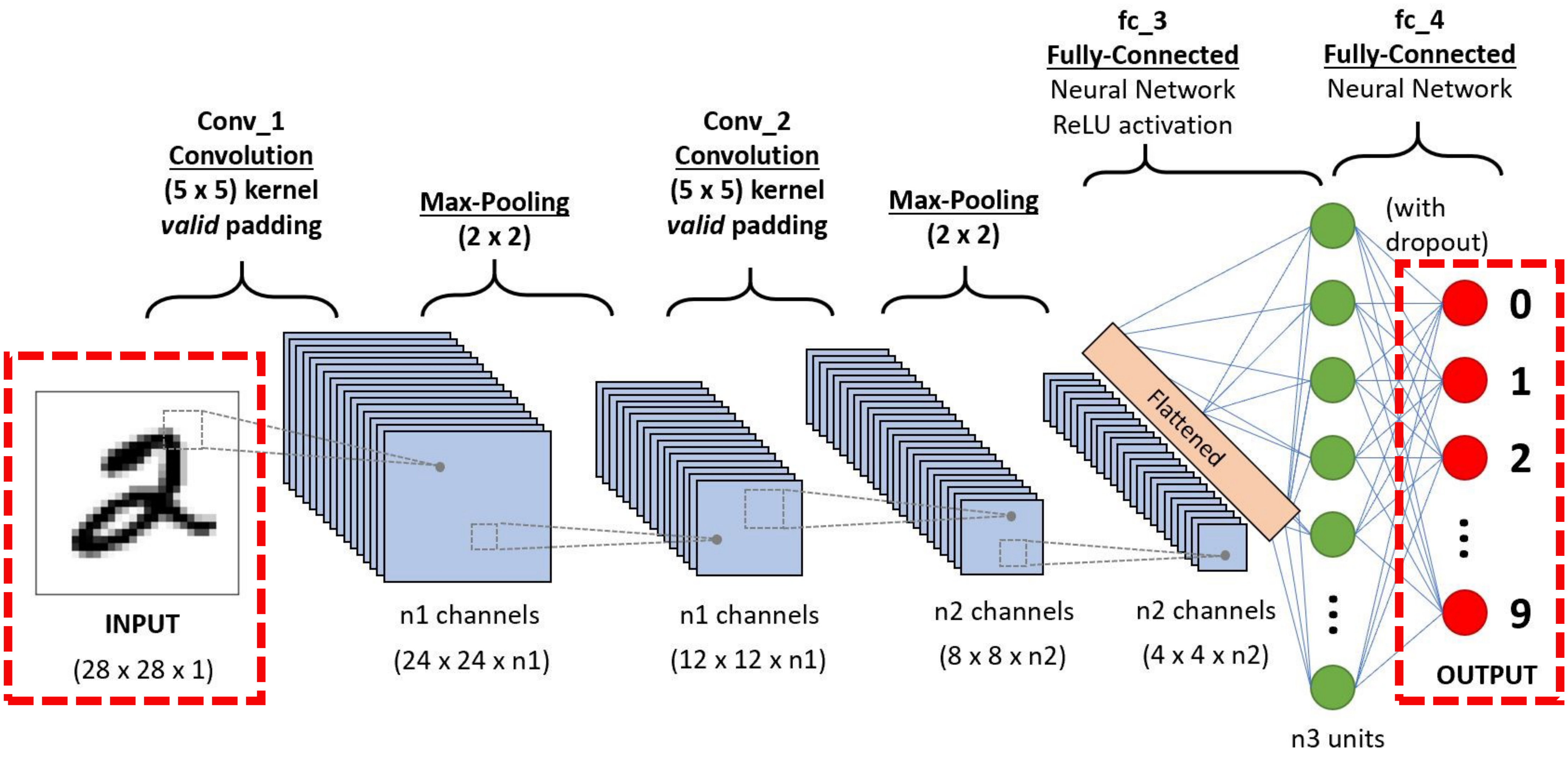
Input

CNN
Model

Output

(Prediction)		(Ground truth)
0.2	Dog	0
0.7	Cat	1
0.1	Bird	0

CNN - Architecture

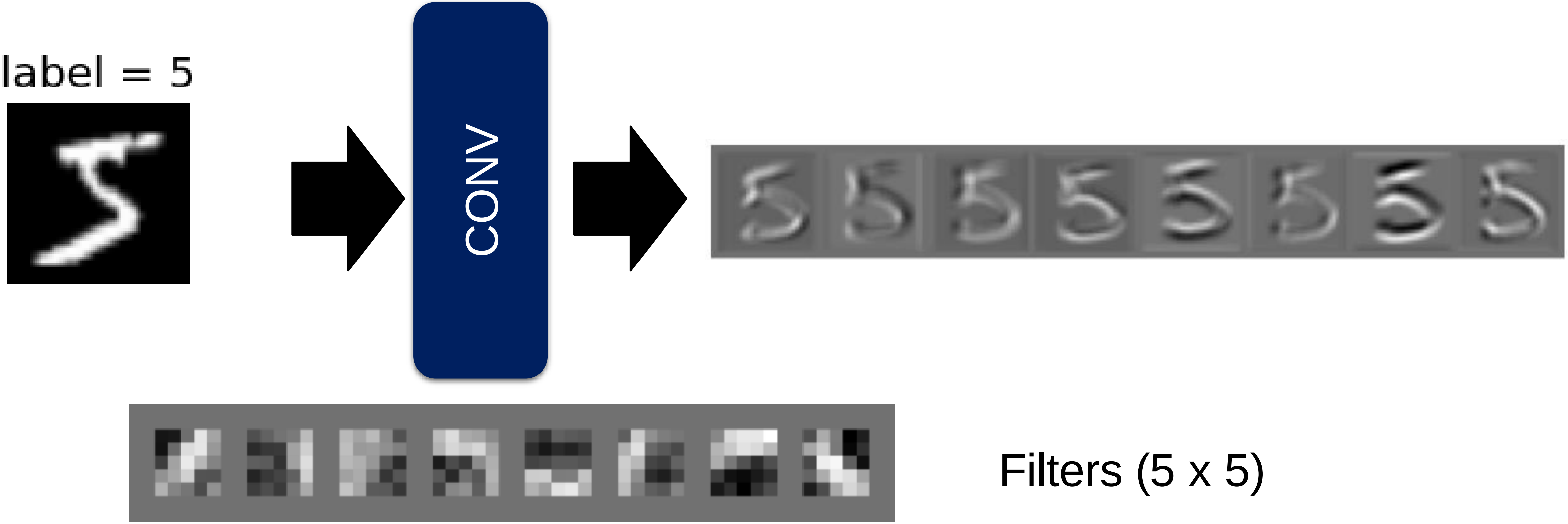


CNN - Components

- Convolution
- Activation Function
- Pooling
- Fully Connected

Convolution

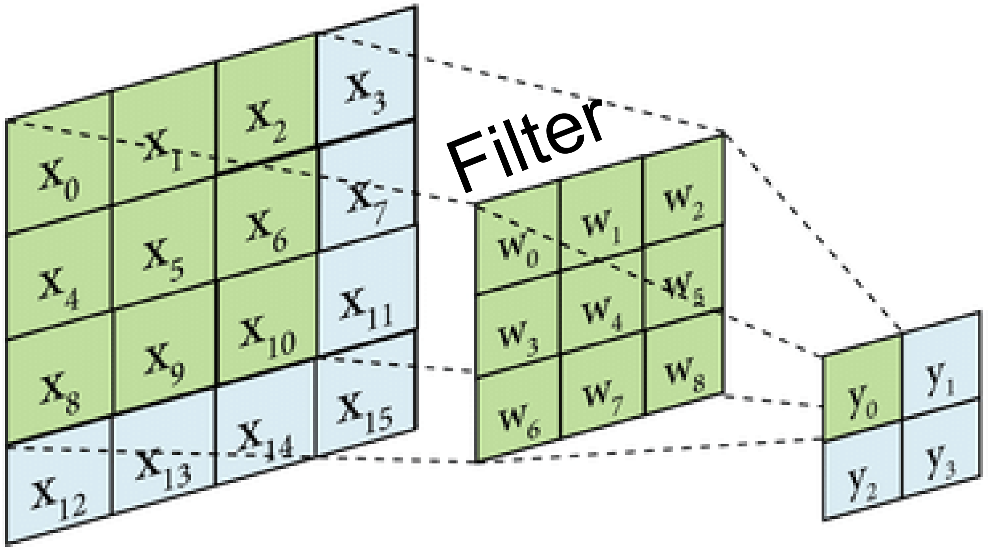
- Filters are used for extracting features in an image.



Filters (5 x 5)

Convolution

- It operates as a sliding window that sweep across the whole image detecting features.



Convolution

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	0	0	0

6 x 6 binary image

-1	-1	1
-1	1	-1
1	-1	-1

3x3 Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

3x3 Filter 2

⋮

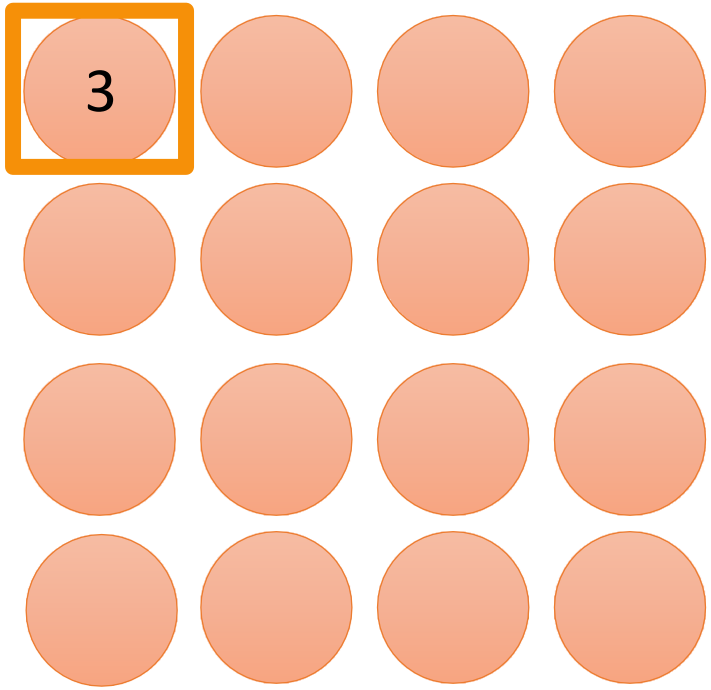
Convolution

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	0	0	0

6 x 6 binary image

-1	-1	1
-1	1	-1
1	-1	-1

Filter 1



Feature Map

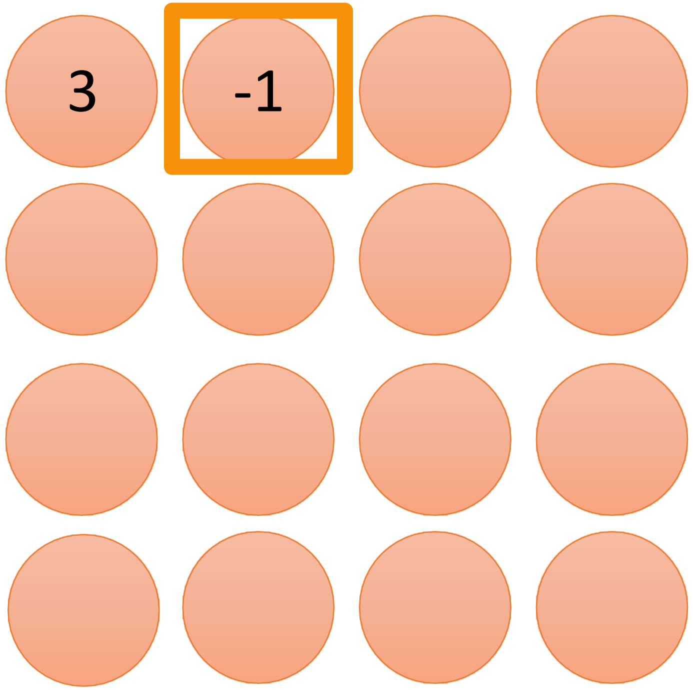
Convolution

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	0	0	0

6 x 6 binary image

-1	-1	1
-1	1	-1
1	-1	-1

Filter 1



Feature Map

Convolution

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	0	0	0

6 x 6 binary image

-1	-1	1
-1	1	-1
1	-1	-1

Filter 1

3	-1	$\boxed{-3}$	

Feature Map

Convolution

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	0	0	0

6 x 6 binary image

-1	-1	1
-1	1	-1
1	-1	-1

Filter 1

3	-1	-3	-1

Feature Map

Convolution

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	0	0	0

6 x 6 binary image

-1	-1	1
-1	1	-1
1	-1	-1

Filter 1

3	-1	-3	-1
-3			

Feature Map

Convolution

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	0	0	0

6 x 6 binary image

-1	-1	1
-1	1	-1
1	-1	-1

Filter 1

3	-1	-3	-1
-3	0	0	-3
-1	-1	-2	3

Feature Map

Convolution

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	0	0	0

6 x 6 binary image

-1	-1	1
-1	1	-1
1	-1	-1

Filter 1

3	-1	-3	-1
-3	0	0	-3
-1	-1	-2	3
-1	-1	2	-2

Feature Map

Convolution

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	0	0	0

6 x 6 binary image

-1	-1	1
-1	1	-1
1	-1	-1

Filter 1

3	-1	-3	-1
-3	0	0	-3
-1	-1	-2	3
-1	-1	2	-2

Feature Map

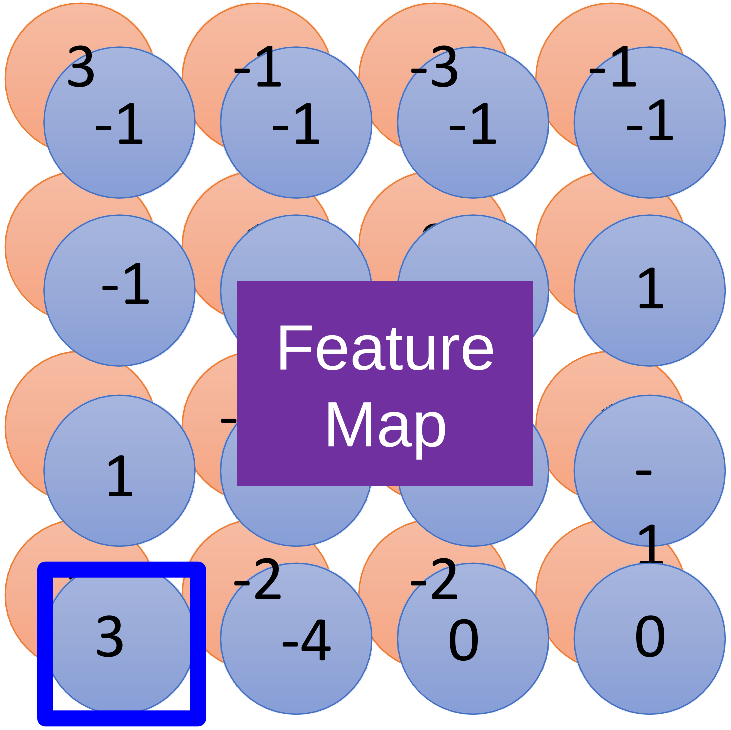
Convolution

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	0	0	0

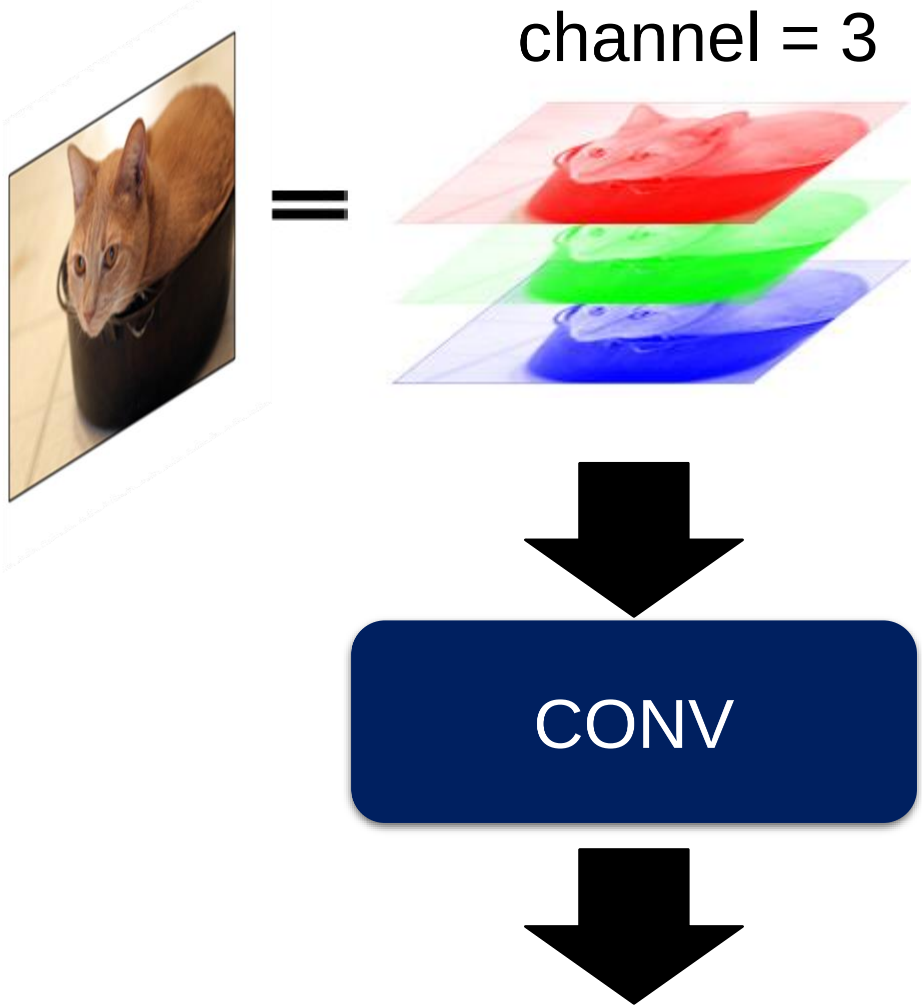
6 x 6 binary image

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2



Convolution



-1	-1	1
-1	1	-1
1	-1	-1

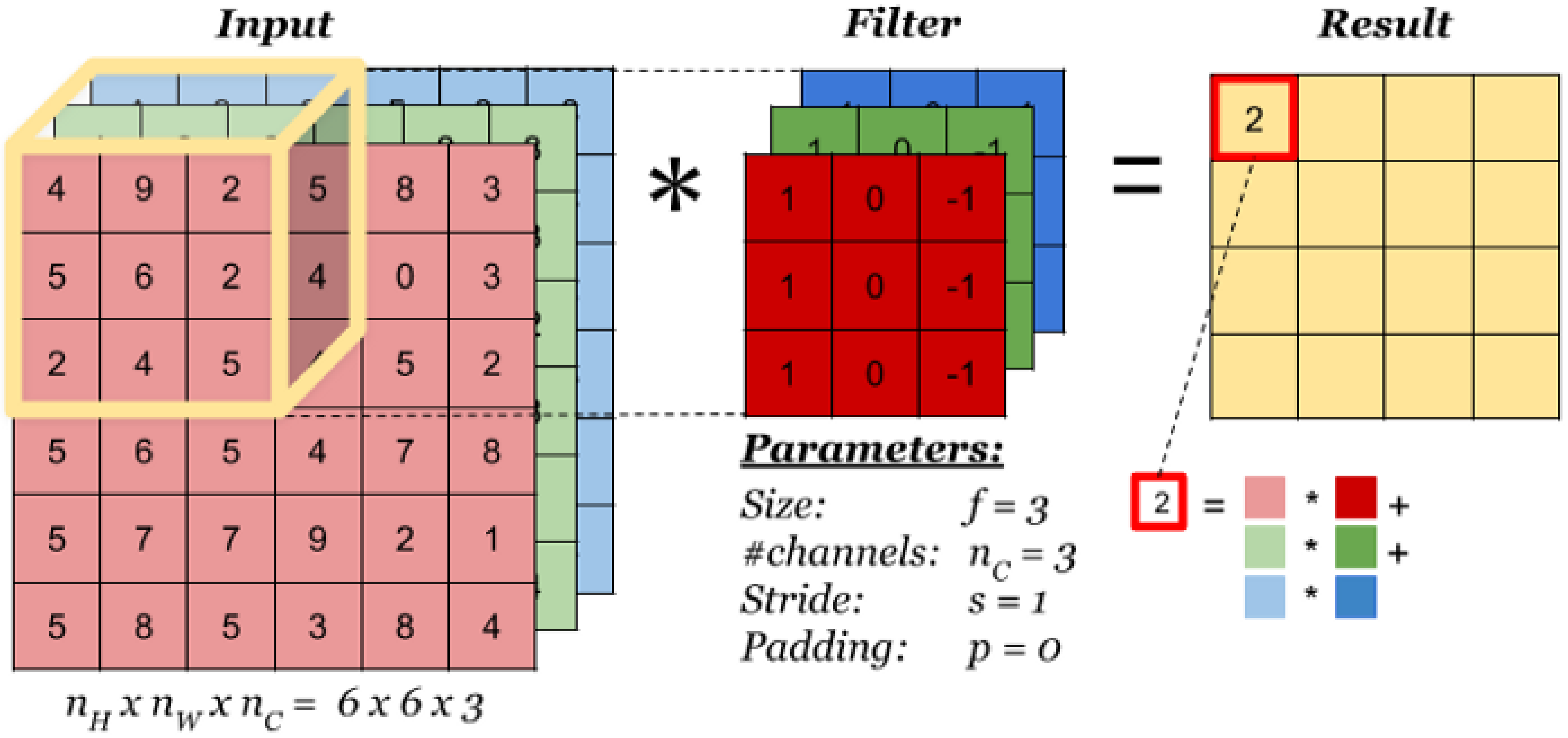
Filter 1
3 x 3 x #channel

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2
3 x 3 x #channel

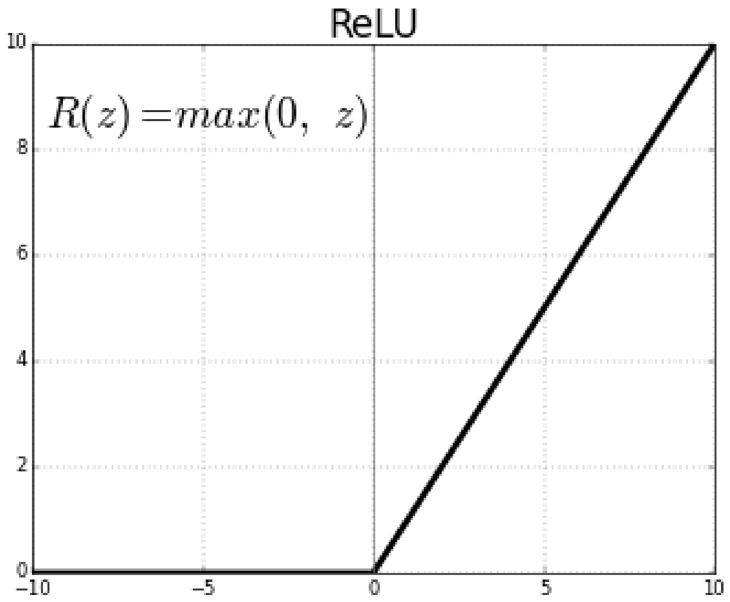
Convolution

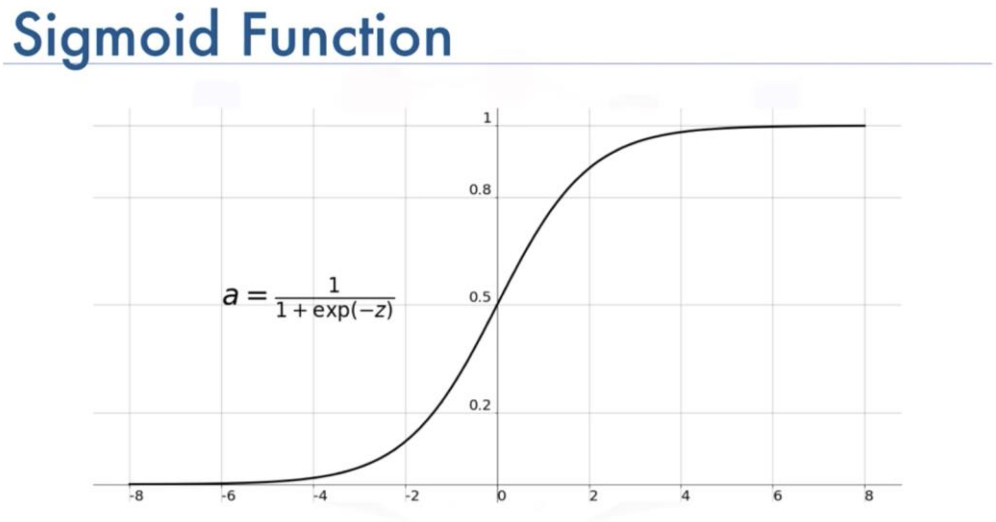
- Convolution is the first layer to extract features from an input image



Activation Function

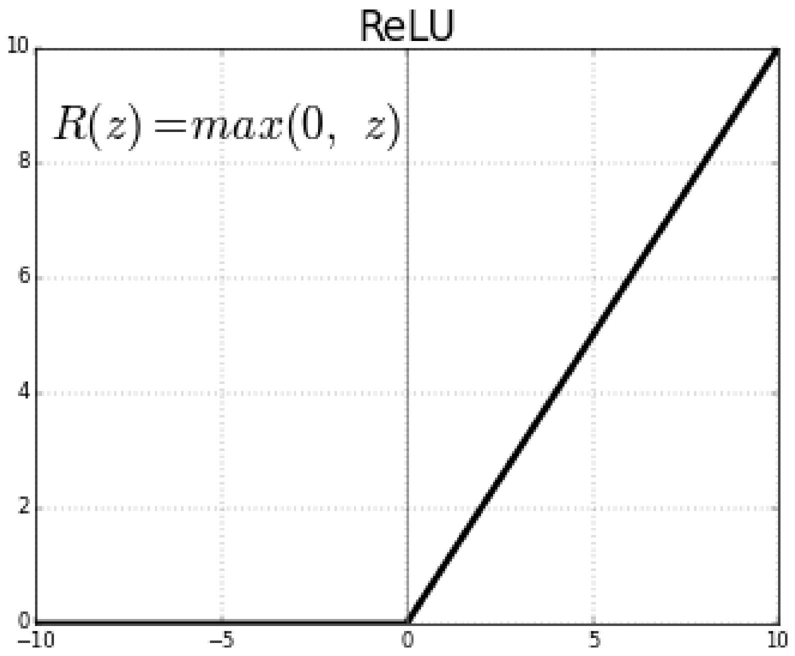
- Activation functions determine the output of current layer (input of the next layer)
- Commonly used activation functions include ReLU, Tanh, Sigmoid and Softmax.





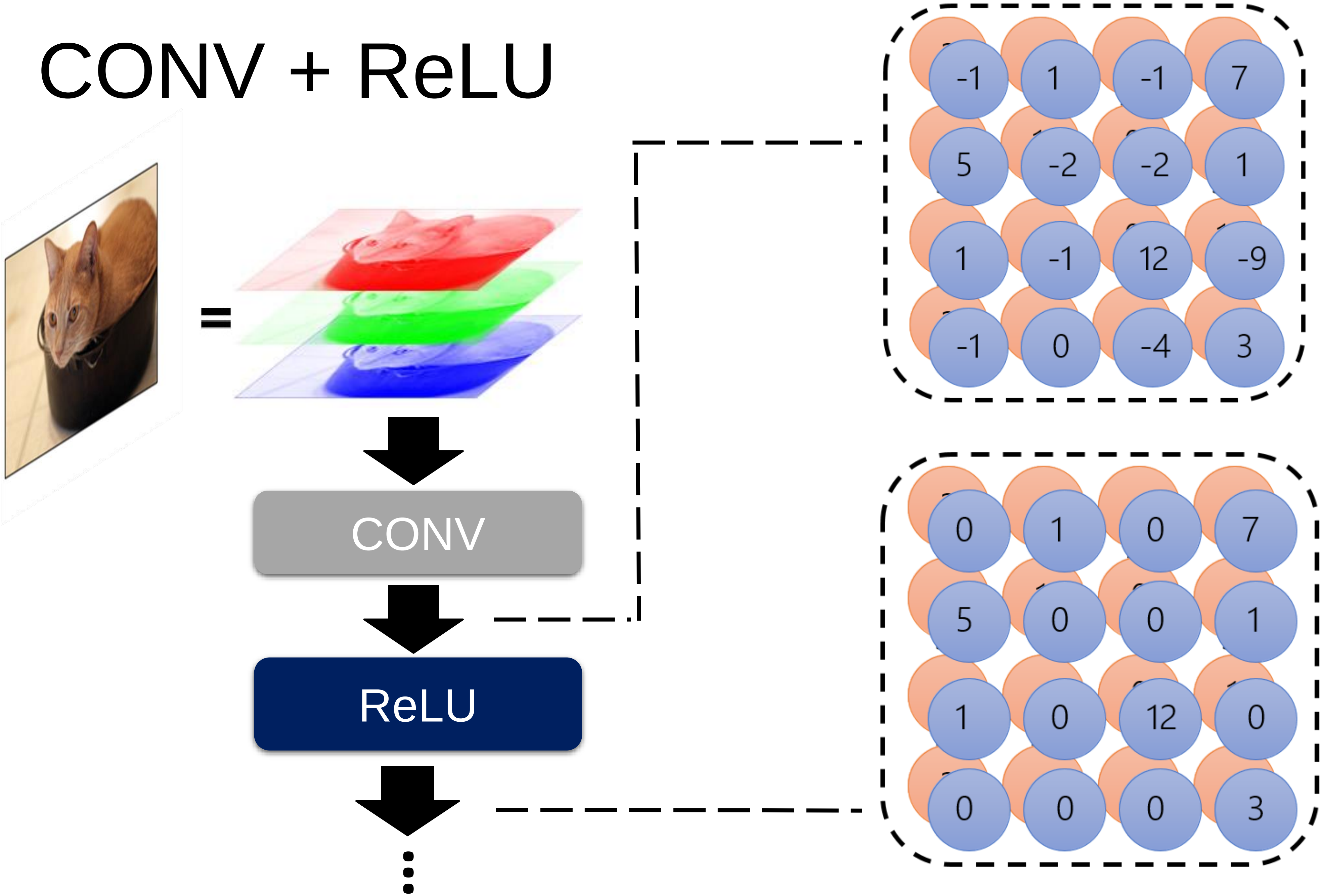
Example – ReLU

15	20	**-10**	35
18	**-11**	25	99
20	**-15**	25	**-10**
11	75	18	23



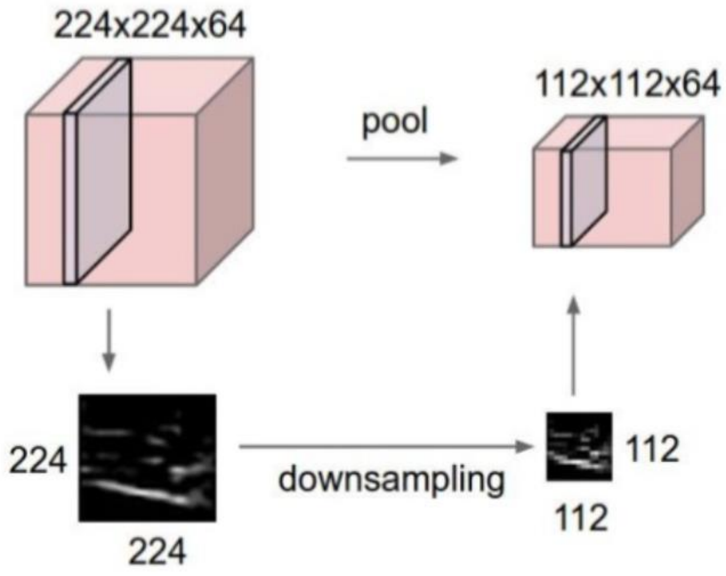
15	20	**0**	35
18	**0**	25	99
20	**0**	25	**0**
11	75	18	23

CONV + ReLU



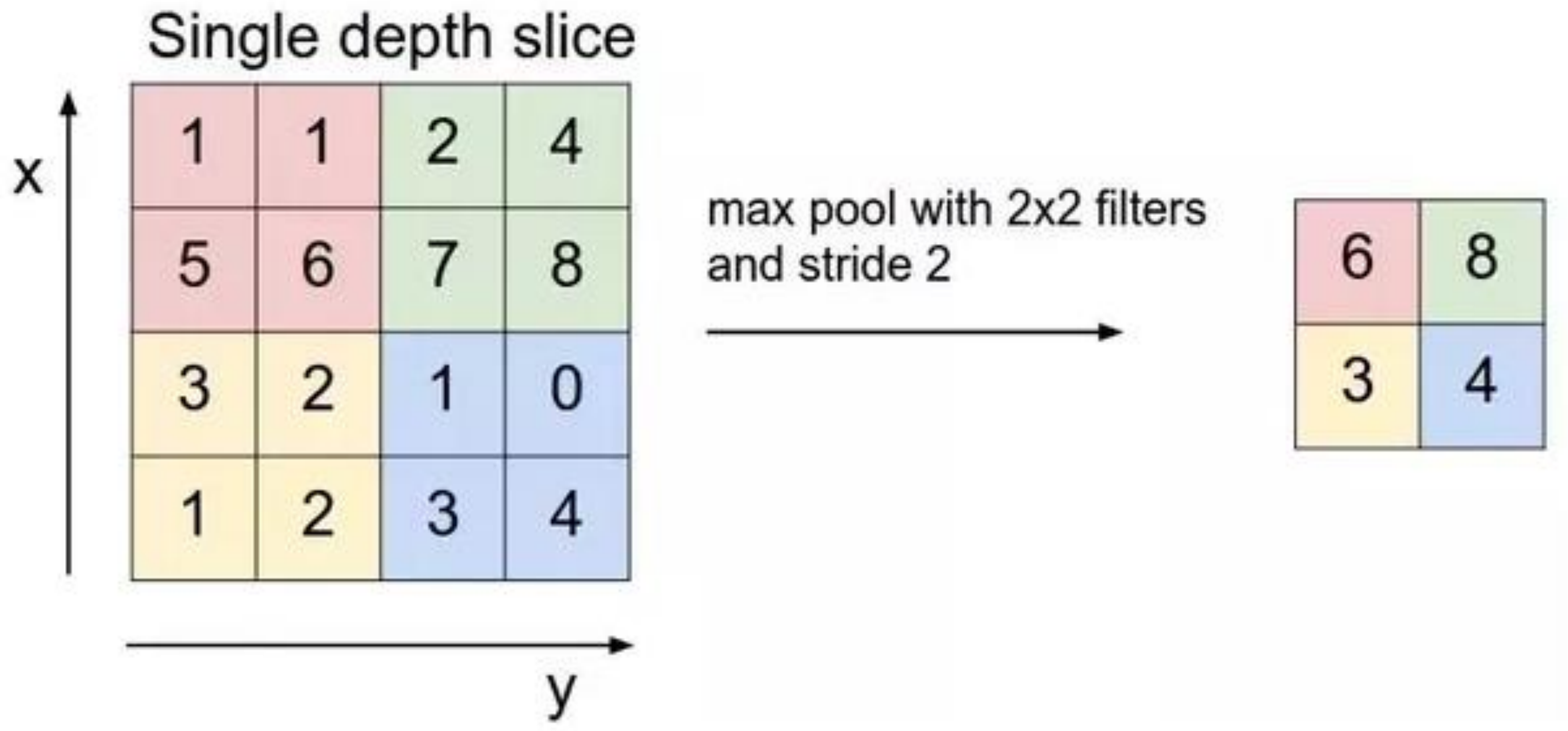
Pooling

- The goal of a pooling layer is to produce a summary statistic of its input and to reduce the spatial dimensions of the feature map (hopefully without losing essential information).
- The three types of pooling operations are Max pooling, Min pooling, and Average pooling.

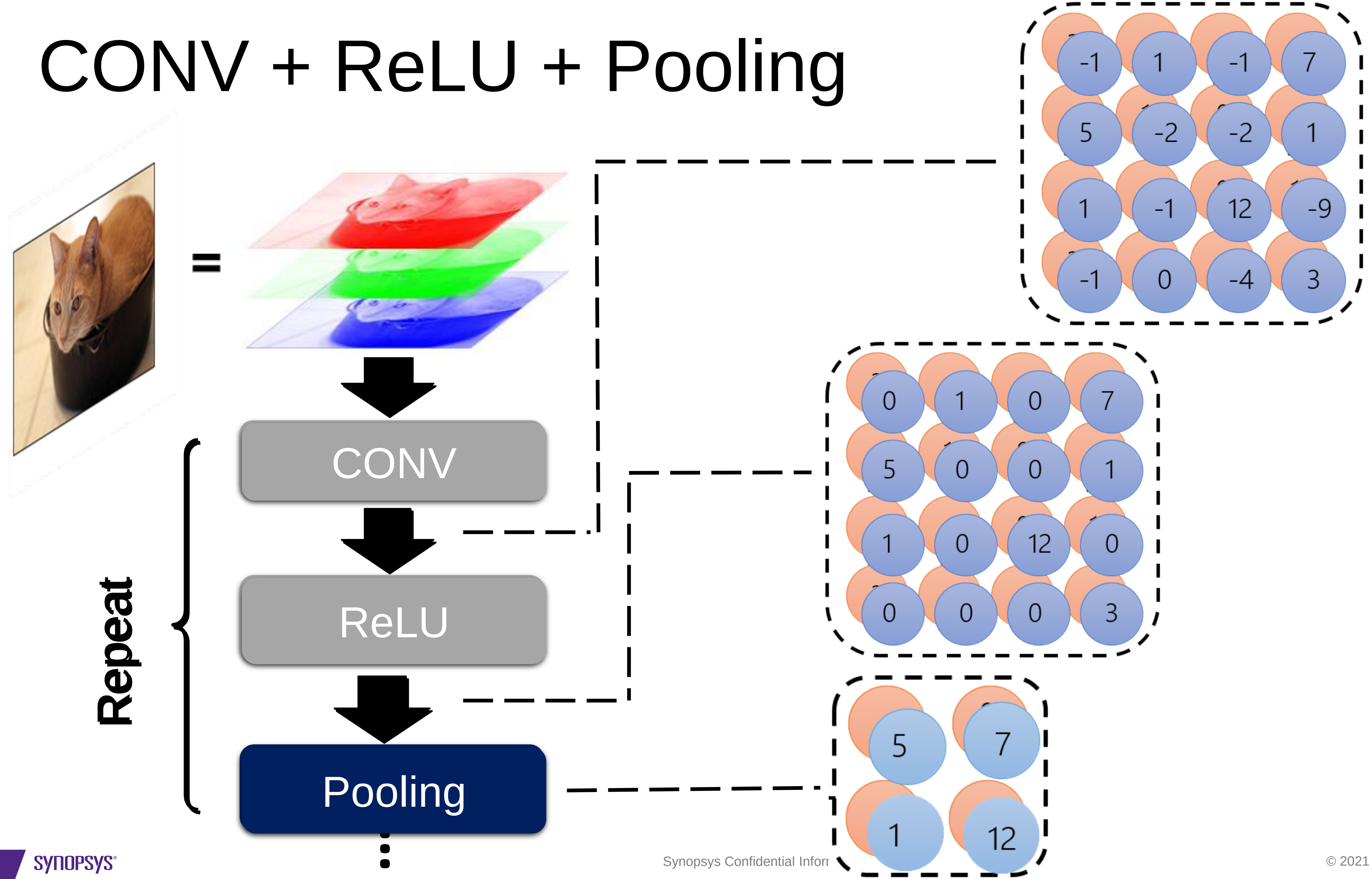


Example – Max pooling

- Max pooling extracts only the maximum activation in each block.

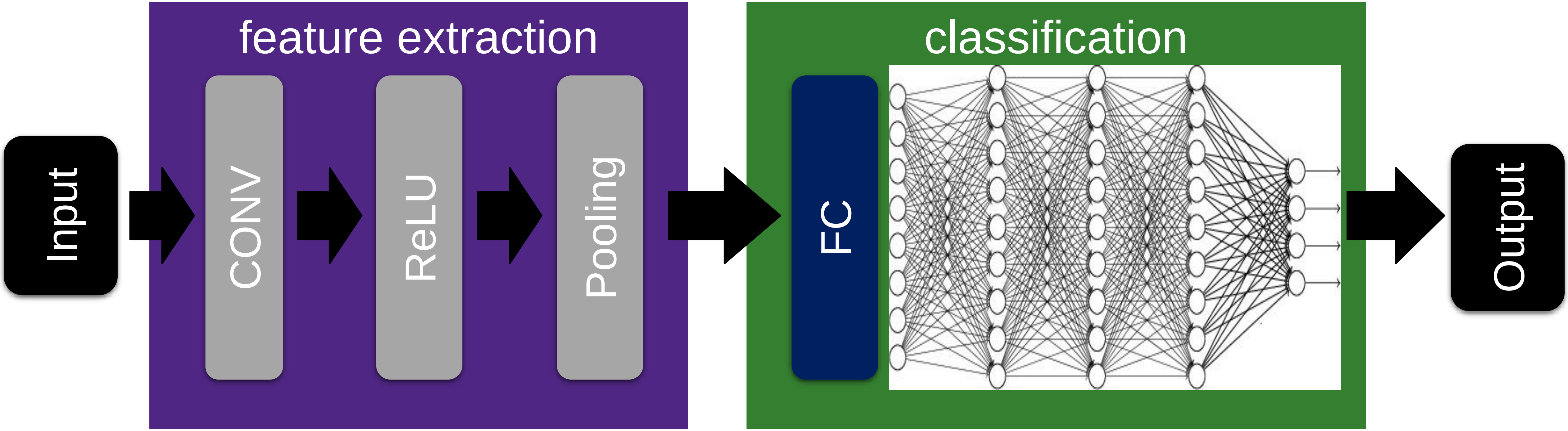


CONV + ReLU + Pooling

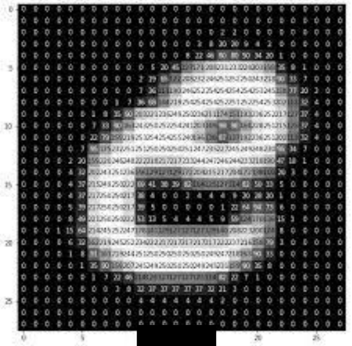


Fully Connected

- After feature extraction, we need to classify the data into various classes. This can be done by using a fully connected (FC) neural network.



CNN - Overall



CONV

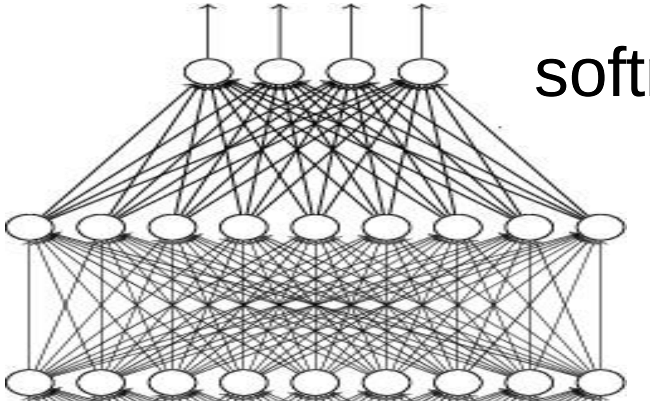
BatchNorm

ReLU

Max pooling

Flatten

... A, B, C, D, ...



softmax

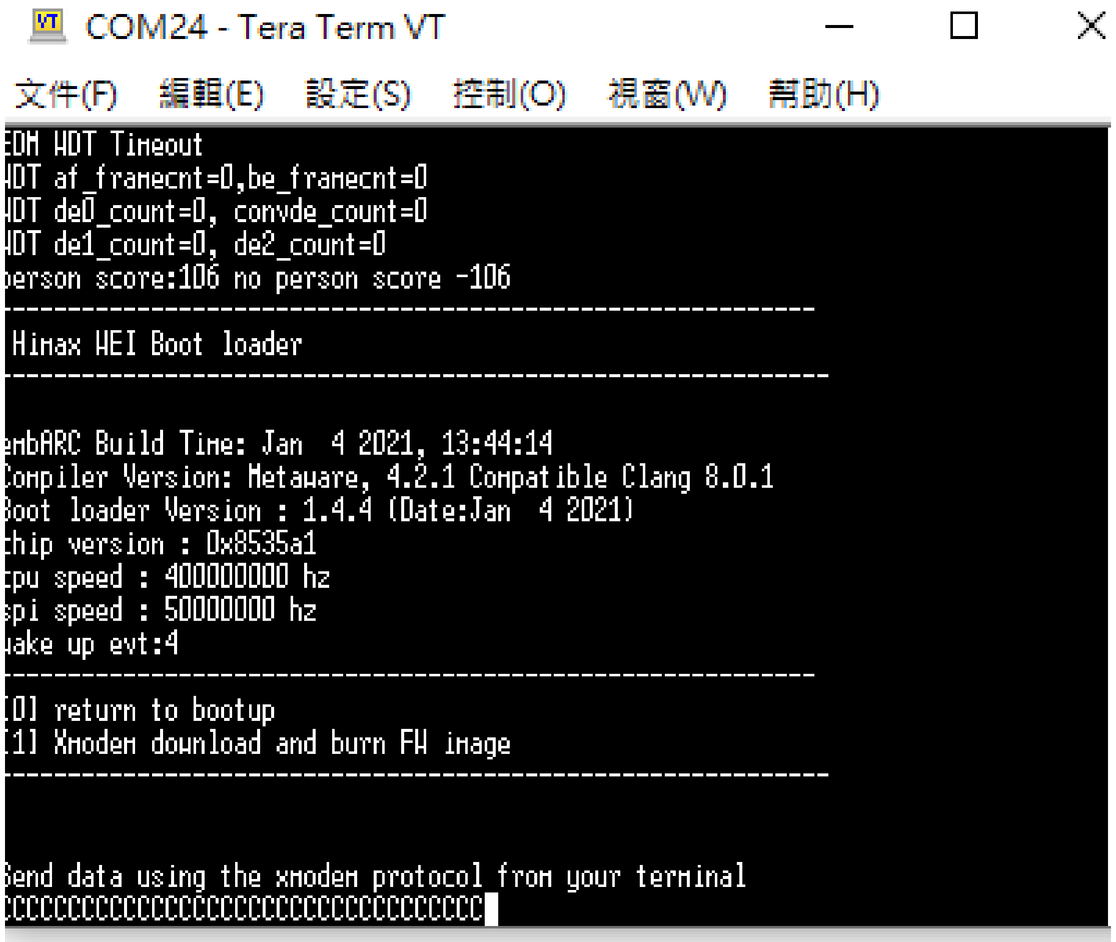
Reference

- Hung-yi Lee, 卷積神經網路 (Convolutional Neural Networks, CNN): https://youtu.be/OP5HcXJg2Aw
- CNN Architecture Image: https://editor.analyticsvidhya.com/uploads/90650dnn2.jpeg
- https://medium.com/daai/%E5%93%87-convolution-neural-network-%E5%8D%B7%E7%A9%8D%E7%A5%9E%E7%B6%93%E7%B6%B2%E7%B5%A1-%E9%80%99%E9%BA%BC%E7%89%B9%E5%88%A5-36d02ce8b5fe
- Max pooling vs min pooling vs average pooling https://medium.com/@bdhuma/which-pooling-method-is-better-maxpooling-vs-minpooling-vs-average-pooling-95fb03f45a9

Hands-on (Lab 4): TensorFlow Lite Example Project Person Detection

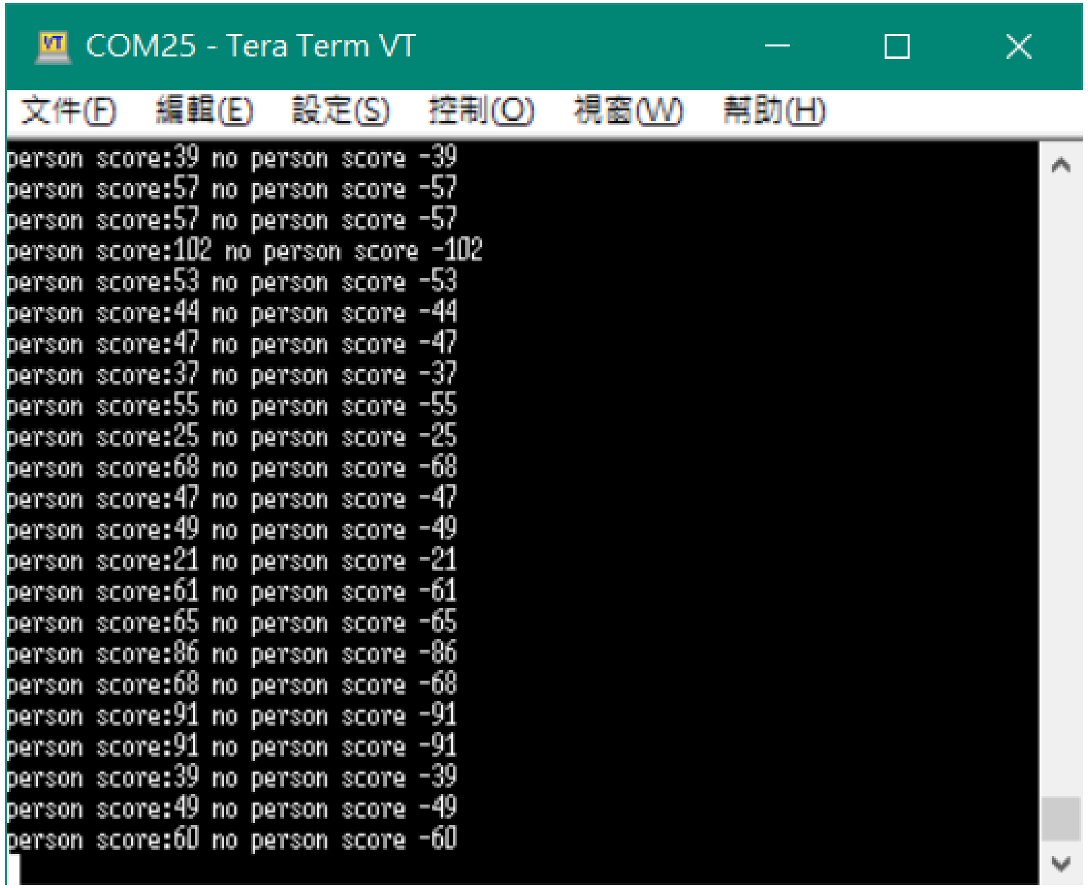
Lab4: Person Detection

- Push WE-I reset button, and download image file to WE-I



Lab4: Person Detection

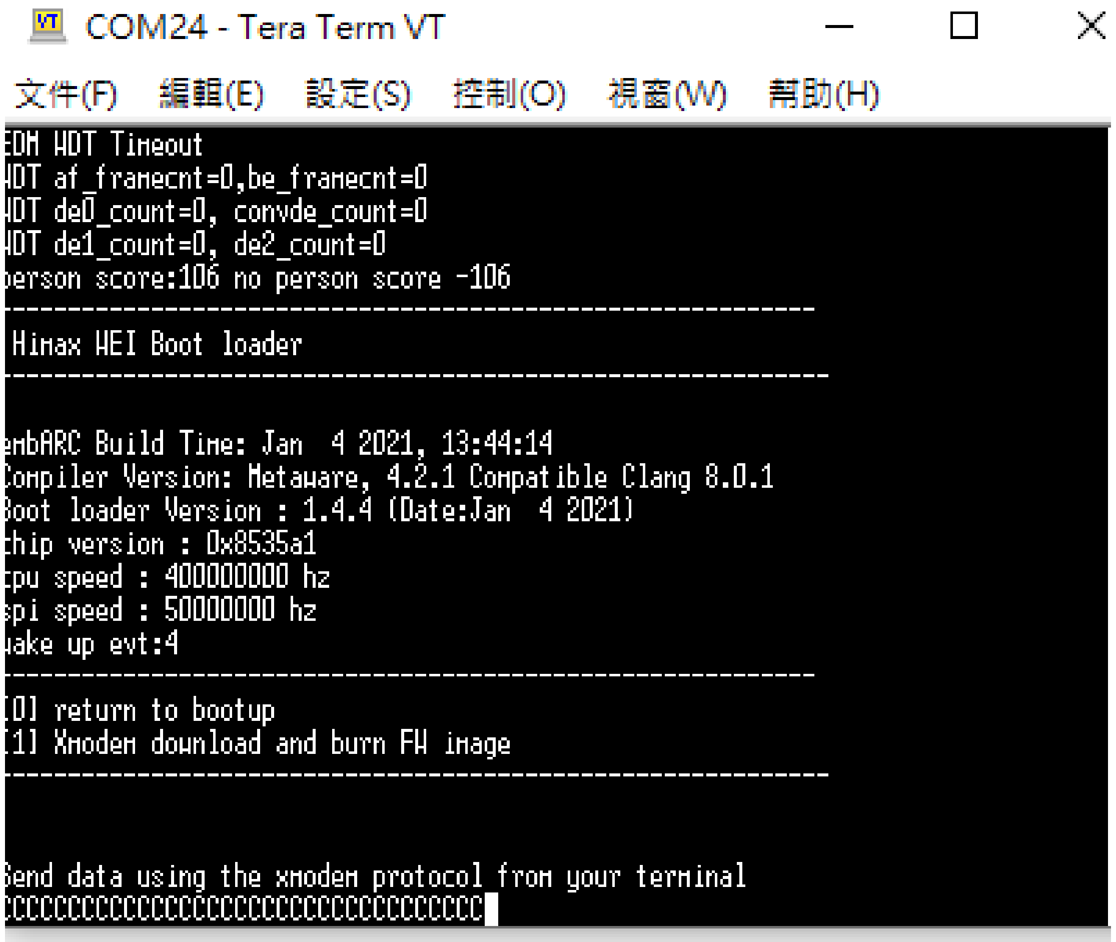
- This example project will detect person by camera.
- During person score > no person score, green LED will turn on.



Hands-on (Lab 4): TensorFlow Lite Example Project Hand-Writing Number Recognition

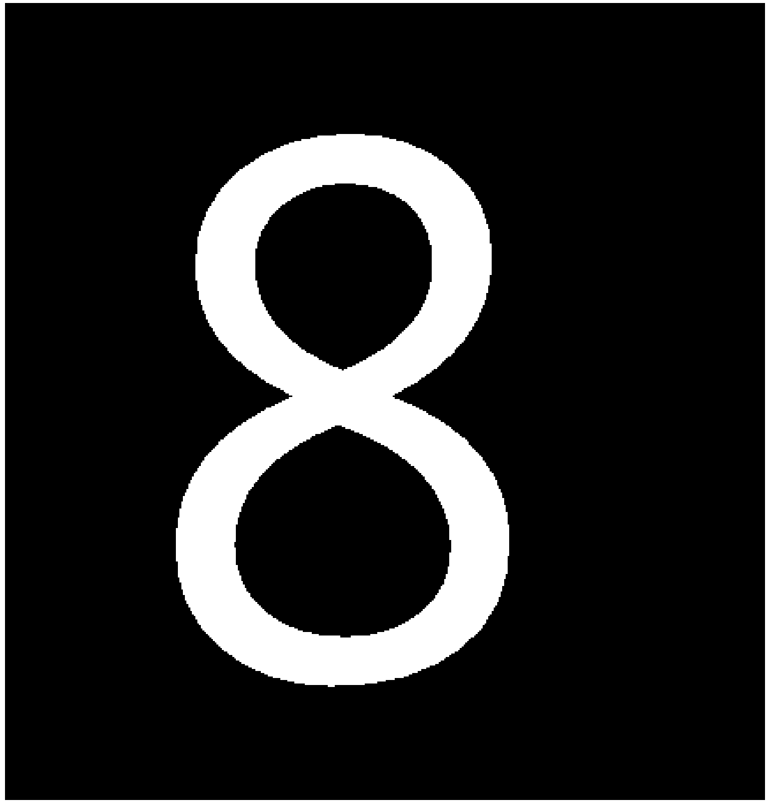
Lab4: Hand-Writing Number Recognition

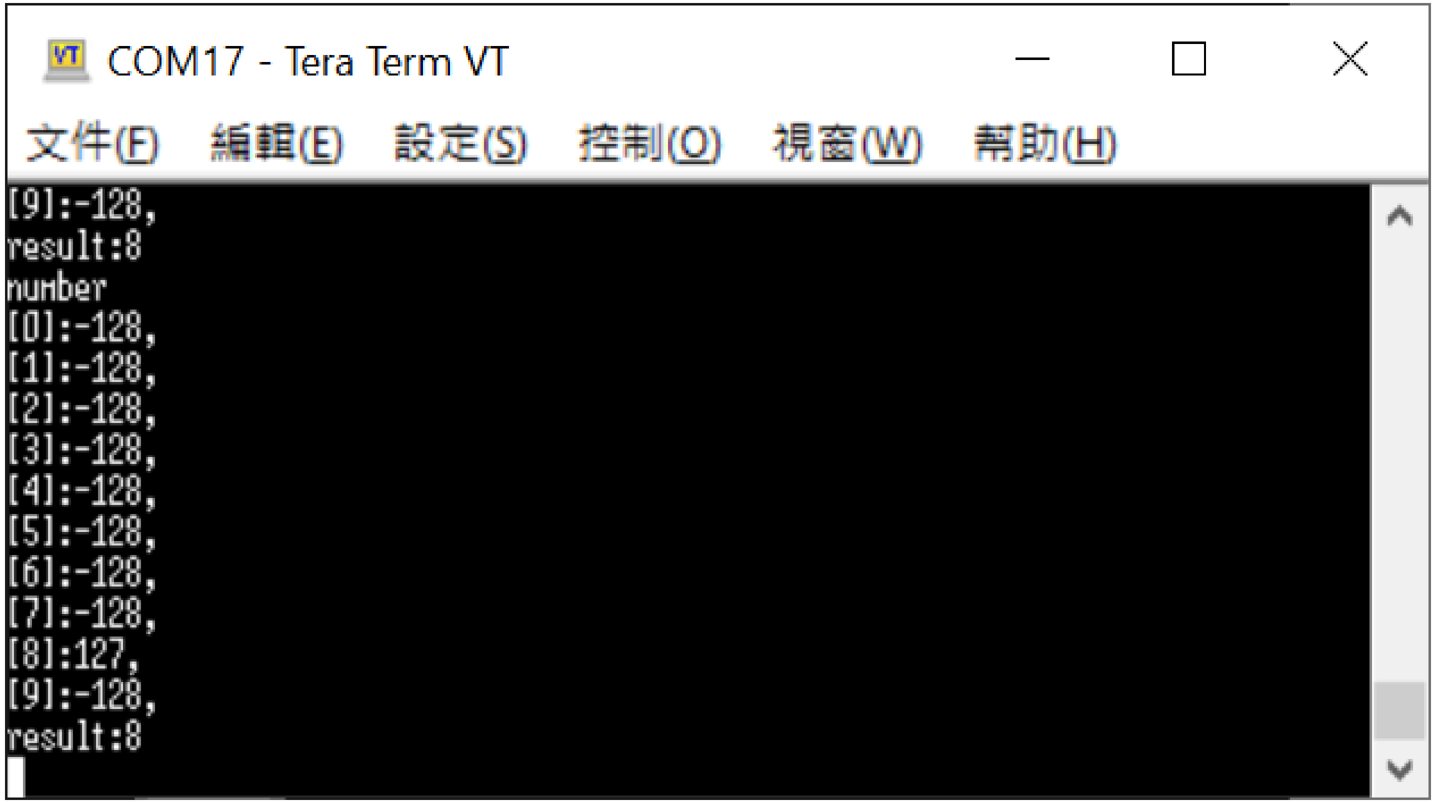
- Push WE-I reset button, and download image file to WE-I



Lab4: Hand-Writing Number Recognition

- This example project will recognize number by camera



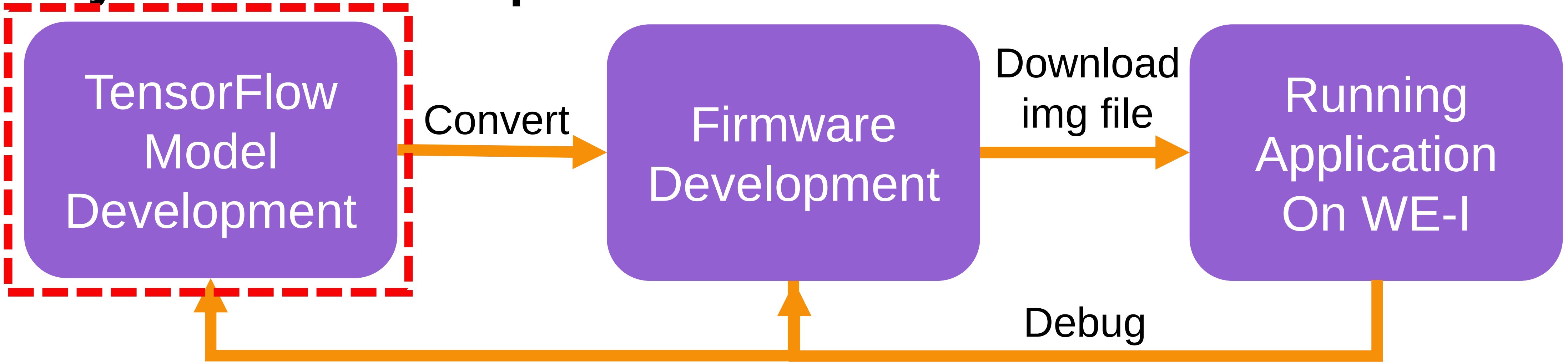


Hands-on (Lab 5-1):
Building Your Own Model with TensorFlow Lite

What is TensorFlow?

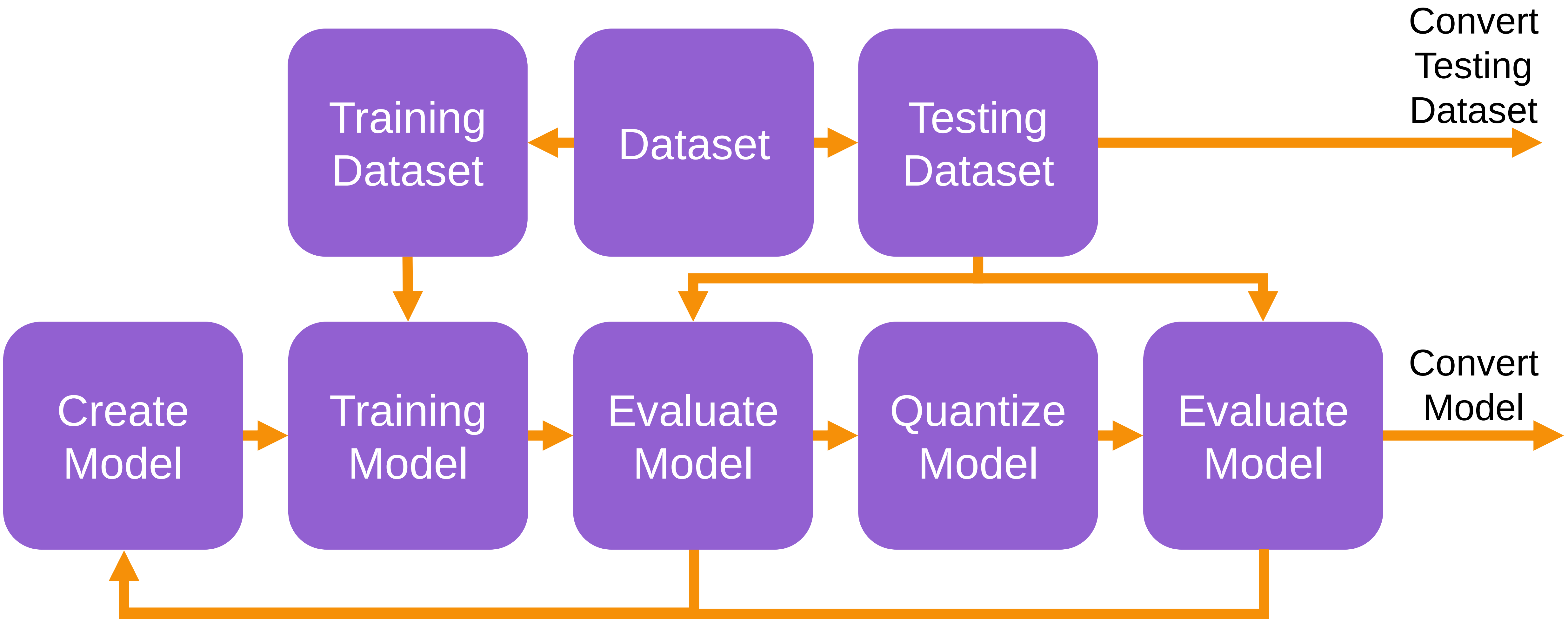
- Created by the Google Brain team, TensorFlow is an open source library for numerical computation and large-scale machine learning.
- TensorFlow bundles together a slew of machine learning and deep learning (aka neural networking) models and algorithms and makes them useful by way of a common metaphor.
- It uses Python to provide a convenient front-end API for building applications with the framework, while executing those applications in high-performance C++.

Project Development Flow

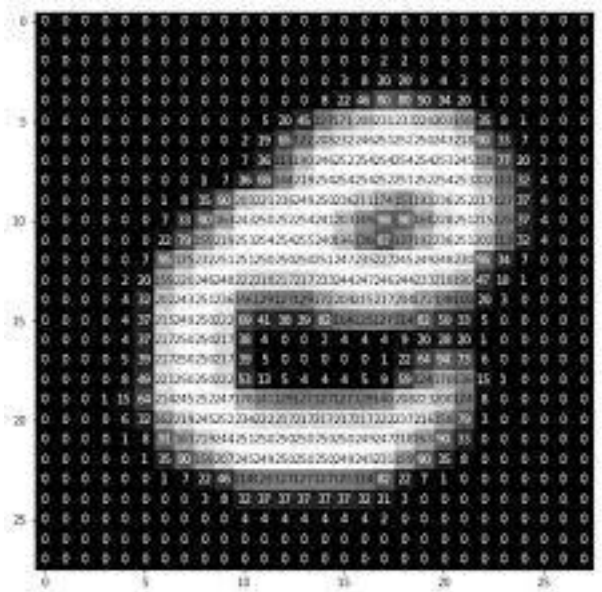


Stage	**TensorFlow Model Development**	**Firmware Development**	**Running Application On WE-I**
Tool	Anaconda Cygwin	Cygwin Metaware or ARC GNU VirtualBox (Ubuntu)	Tera Term USB Micro
Language	Python 3	C language C++ language	

TensorFlow Model Development



Lab5: EMNIST Letters Recognition (Training)



e

Input

Model

Answer

Lab5: EMNIST Letters Recognition (Training)

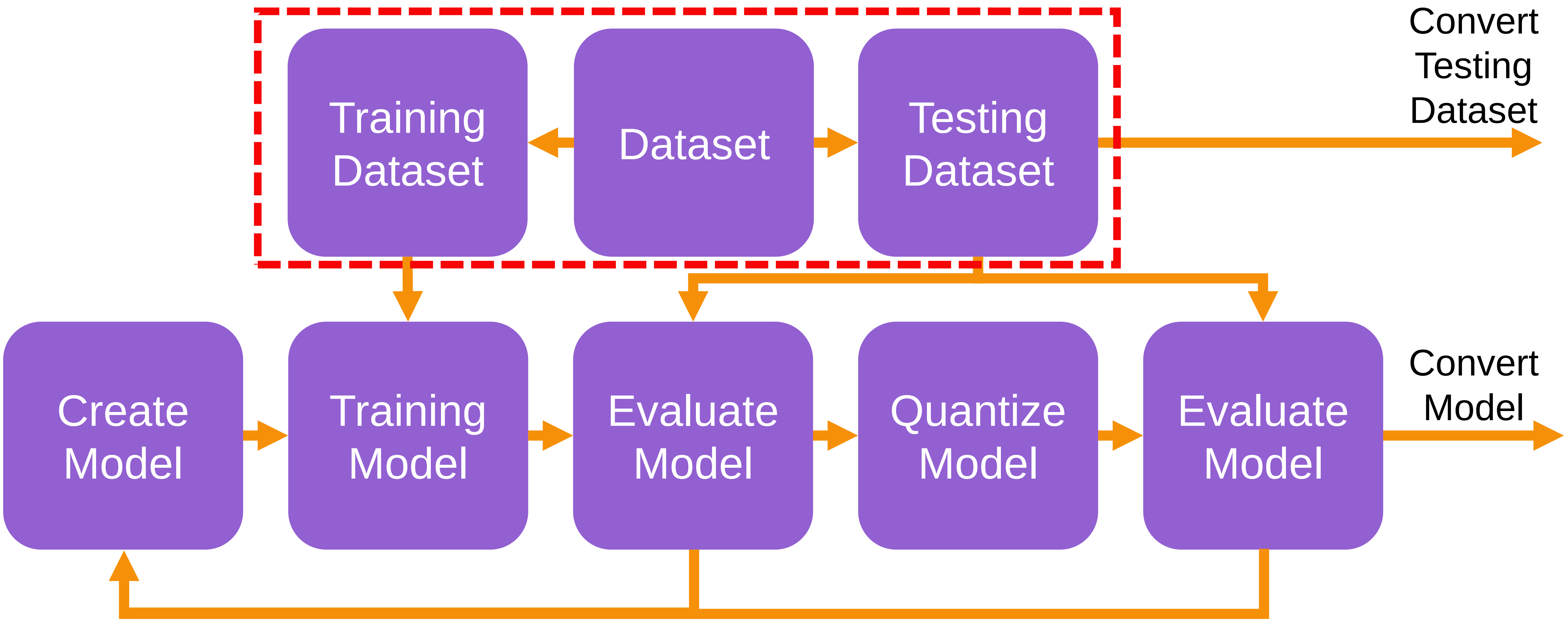
- Open Jupyter Notebook (TensorFlow)
- Go to “Lab5_tflm_emnist_training/”
- Open “Lab5_tflm_emnist_training.ipynb”

Lab5: EMNIST Letters Recognition (Training)

- Import module and function you need to build your model

```
import tensorflow.compat.v2 as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
from emnist import extract_test_samples
from emnist import extract_training_samples
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense
from tensorflow.keras.layers import Activation, BatchNormalization, Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
```

Lab5: EMNIST Letters Recognition (Training)



Lab5: EMNIST Letters Recognition (Training)

- Load dataset

```
# Import training and testing dataset

img_rows = 28
img_cols = 28
num_classes = 26
input_shape = (img_rows, img_cols, 1)
filter_x = 5
filter_y = 5

train_images, train_labels = extract_training_samples('letters')
test_images, test_labels = extract_test_samples('letters')

# Make class numbering start at 0
train_labels = train_labels - 1
test_labels = test_labels - 1
```

→ Emnist Dataset

letter

digits

Lab5: EMNIST Letters Recognition (Training)

- Dataset preprocessing

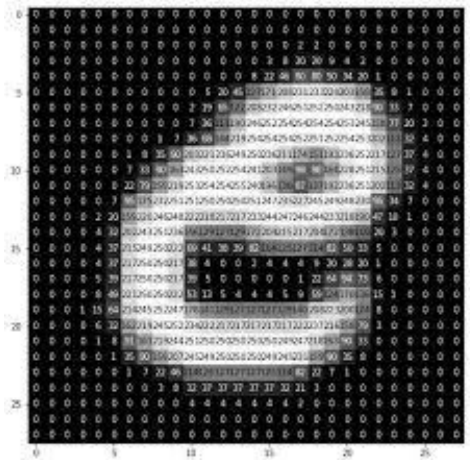
Reshape from 124800*28*28 to 124800*28*28*1

```
# Dataset preprocessing #1

# Reshape
train_images = train_images.reshape(train_images.shape[0], img_rows, img_cols, 1)
test_images = test_images.reshape(test_images.shape[0], img_rows, img_cols, 1)

# Transfer to nparray
train_images = train_images.astype('float32')
test_images = test_images.astype('float32')

train_labels = to_categorical(train_labels, num_classes, dtype = 'float32')
test_labels = to_categorical(test_labels, num_classes, dtype = 'float32')
```

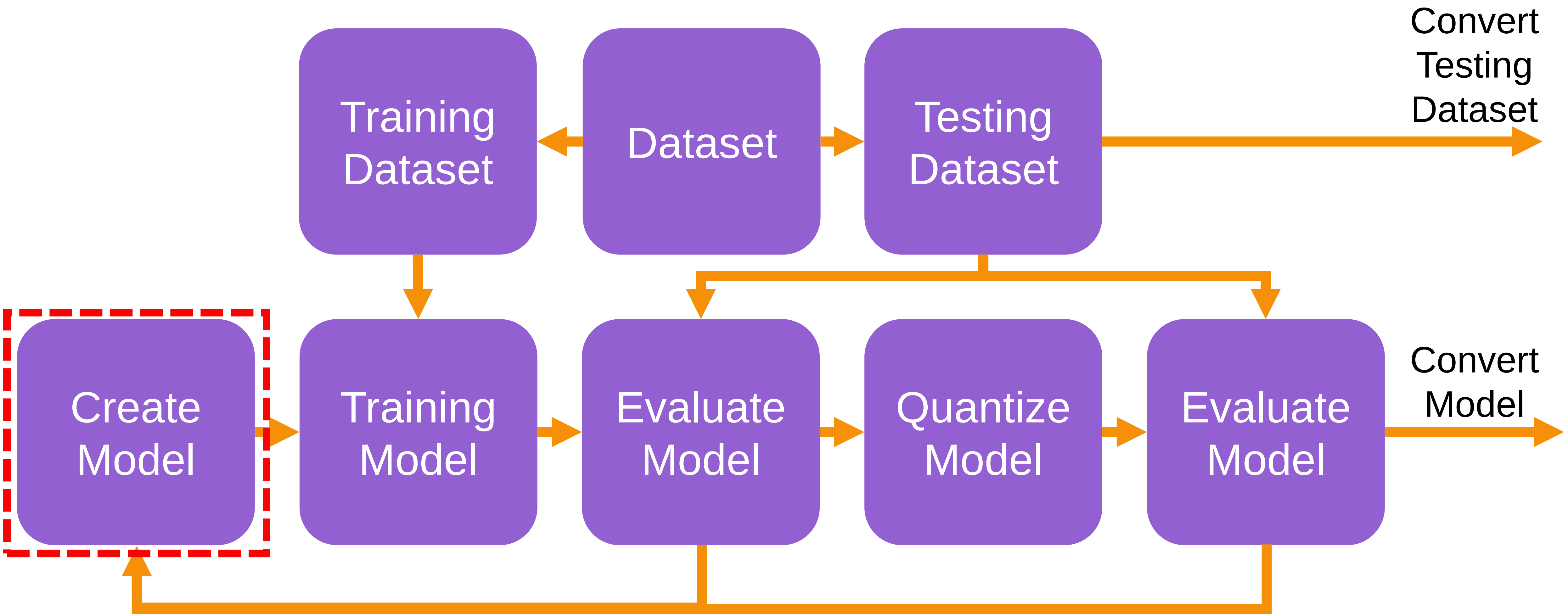
Lab5: EMNIST Letters Recognition (Training)

- Dataset preprocessing

```
# Dataset preprocessing #2(continue)

# Normalize
def thinning(image):
    tmp = np.where(image < 210.0, 0, image)
    return np.where(image < 210.0, 0, 255)

train_images = thinning(train_images)
train_images = (train_images - 128.0) / 128.0

test_images = thinning(test_images)
test_images = (test_images - 128.0) / 128.0
```



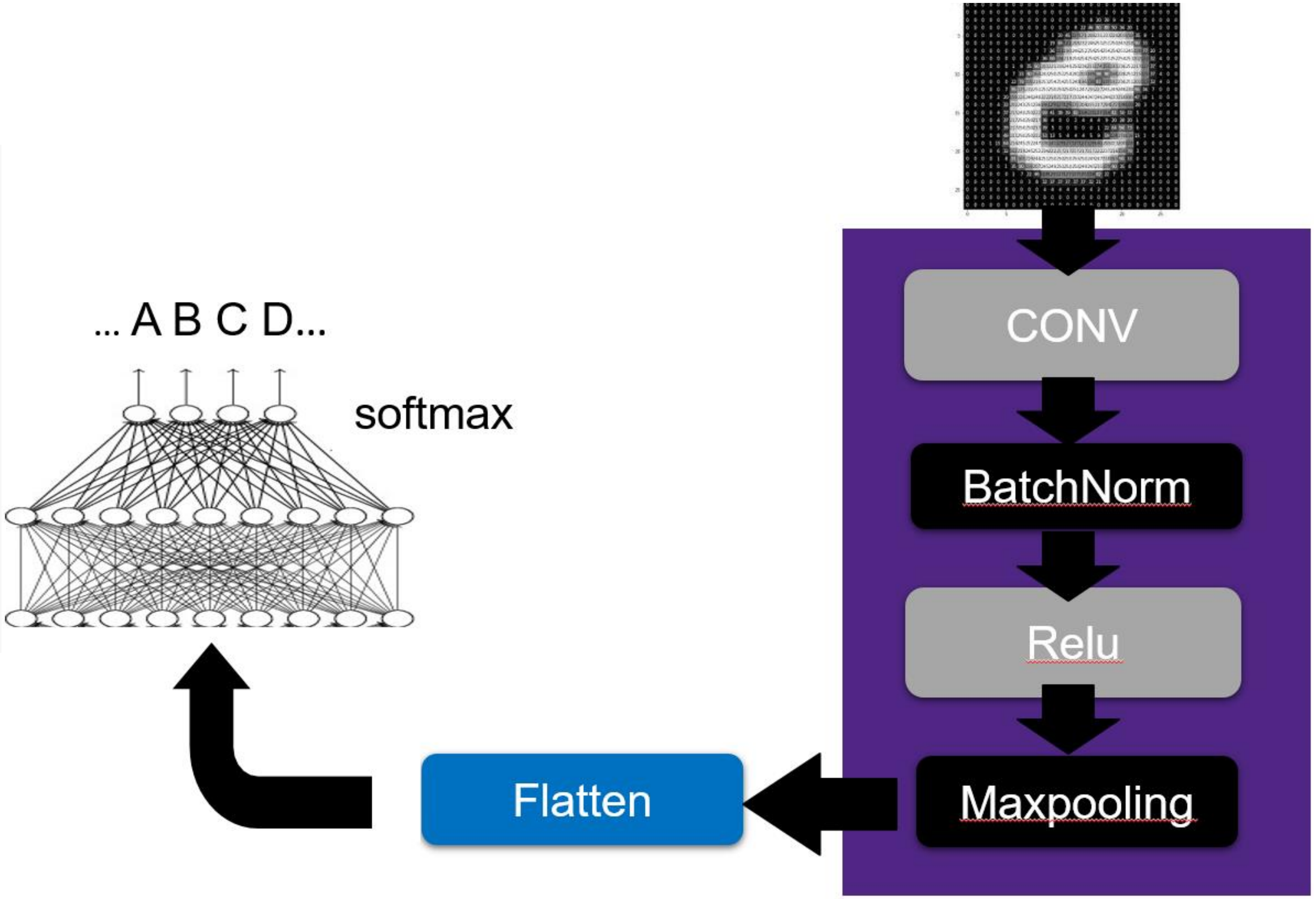
Lab5: EMNIST Letters Recognition (Training)



Lab5: EMNIST Letters Recognition (Training)

- Model Create

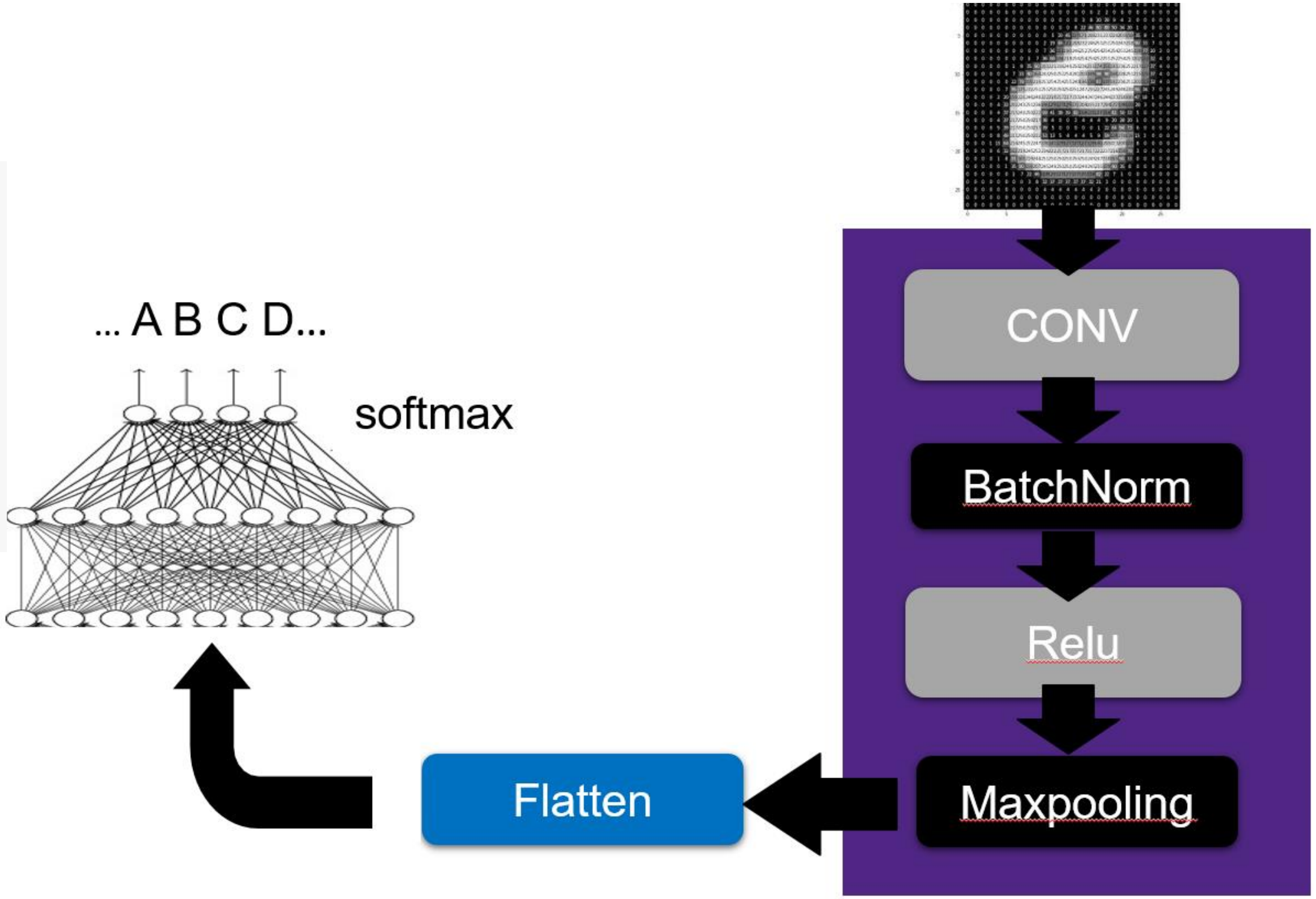
```
# Model create #1

model=Sequential()
#Conv1
model.add(Conv2D(filters=16,
                 kernel_size=(filter_x, filter_y),
                 padding="same",
                 input_shape=input_shape))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(MaxPooling2D())
```



Lab5: EMNIST Letters Recognition (Training)

- Model Create

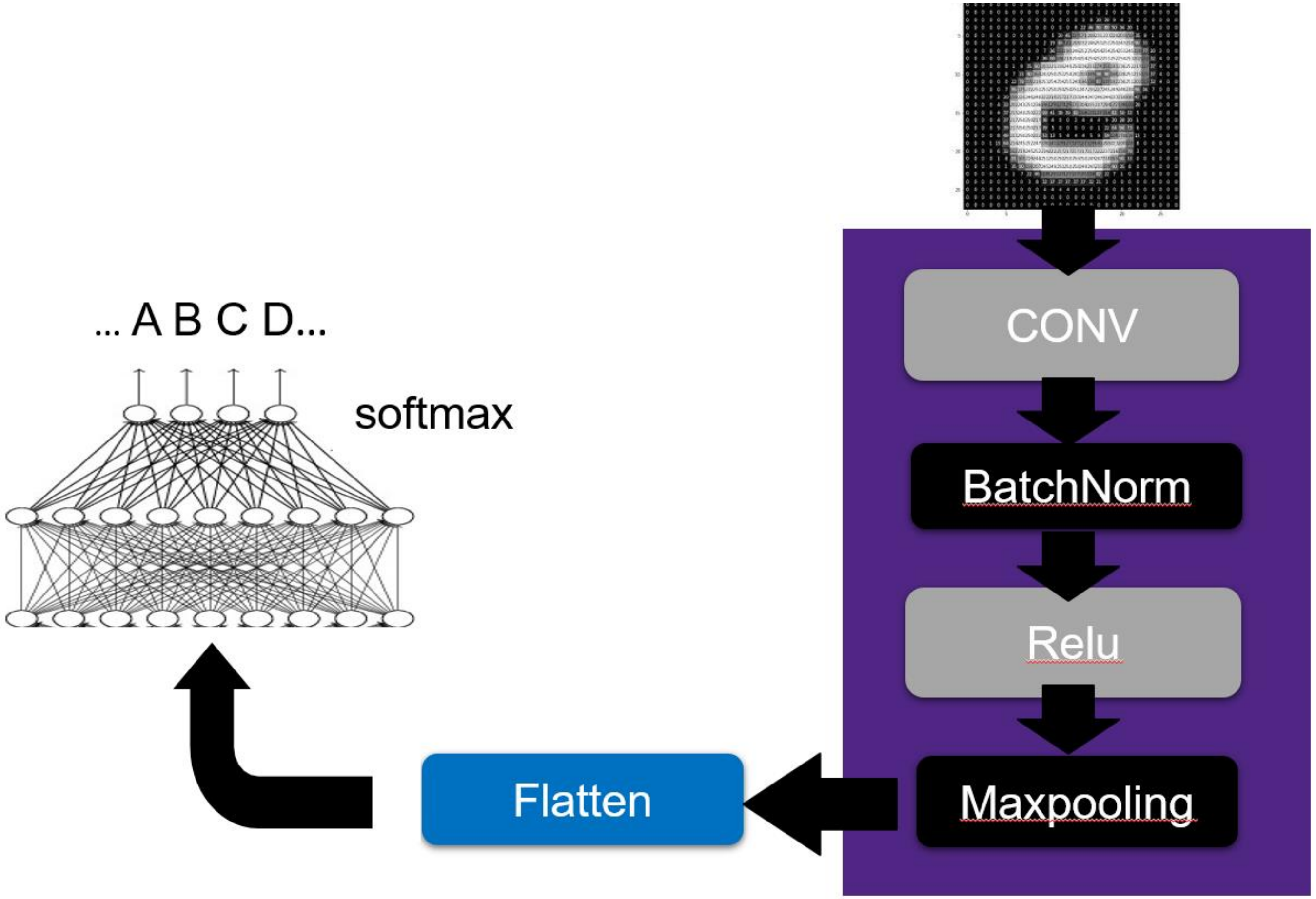
```
#Conv3
model.add(Conv2D(filters=32,
                 kernel_size=(filter_x, filter_y),
                 padding="same",
                 input_shape=input_shape))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(MaxPooling2D())
```



Lab5: EMNIST Letters Recognition (Training)

- Model Create

```
# Model create #2(continue)

#FC1
model.add(Flatten())
model.add(Dense(64))
model.add(BatchNormalization())
model.add(Activation("relu"))

#FC2
model.add(Dense(num_classes))
model.add(Activation("softmax"))
```



Lab5: EMNIST Letters Recognition (Training)

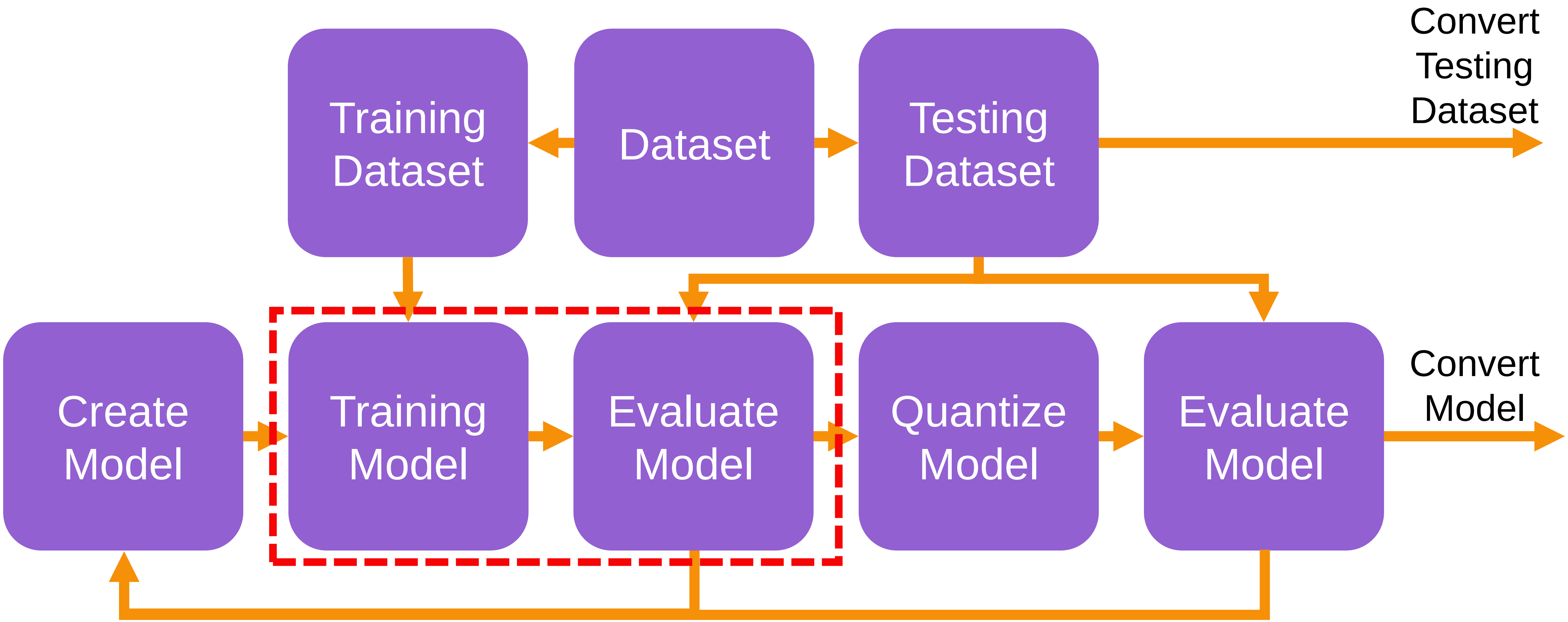
- Show model

```
# Show your model

print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 16)        416
_________________________________________________________________
batch_normalization (BatchNo (None, 28, 28, 16)        64
_________________________________________________________________
activation (Activation)      (None, 28, 28, 16)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 32)        12832
_________________________________________________________________
batch_normalization_1 (Batch (None, 14, 14, 32)        128
```

```
_________________________________
dense_1 (Dense)
_________________________________
activation_4 (Activation)
=================================
Total params: 59,642
Trainable params: 59,354
Non-trainable params: 288
_________________________________
..
```

Lab5: EMNIST Letters Recognition (Training)



Lab5: EMNIST Letters Recognition (Training)

- Model Training

```
# Training model

#Define optimizer loss function and merics
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Set training
model.fit(train_images,train_labels,
          validation_split = 0.2,
          batch_size = 200,
          verbose = 1,
          epochs = 2
          )
```

```
Epoch 1/2
500/500 [==============================] - 120s 240ms/step - loss: 0.6956 - accuracy: 0
cy: 0.8585
Epoch 2/2
 89/500 [====>.........................] - ETA: 1:42 - loss: 0.2940 - accuracy: 0.9103
```

Lab5: EMNIST Letters Recognition (Training)

- Model Evaluation

```
# Model Evaluation
score = model.evaluate(test_images, test_labels, verbose = 0)

print('test loss', score[0])
print('accuracy', score[1])
```

```
test loss 0.2672998011112213
accuracy 0.9141826629638672
```

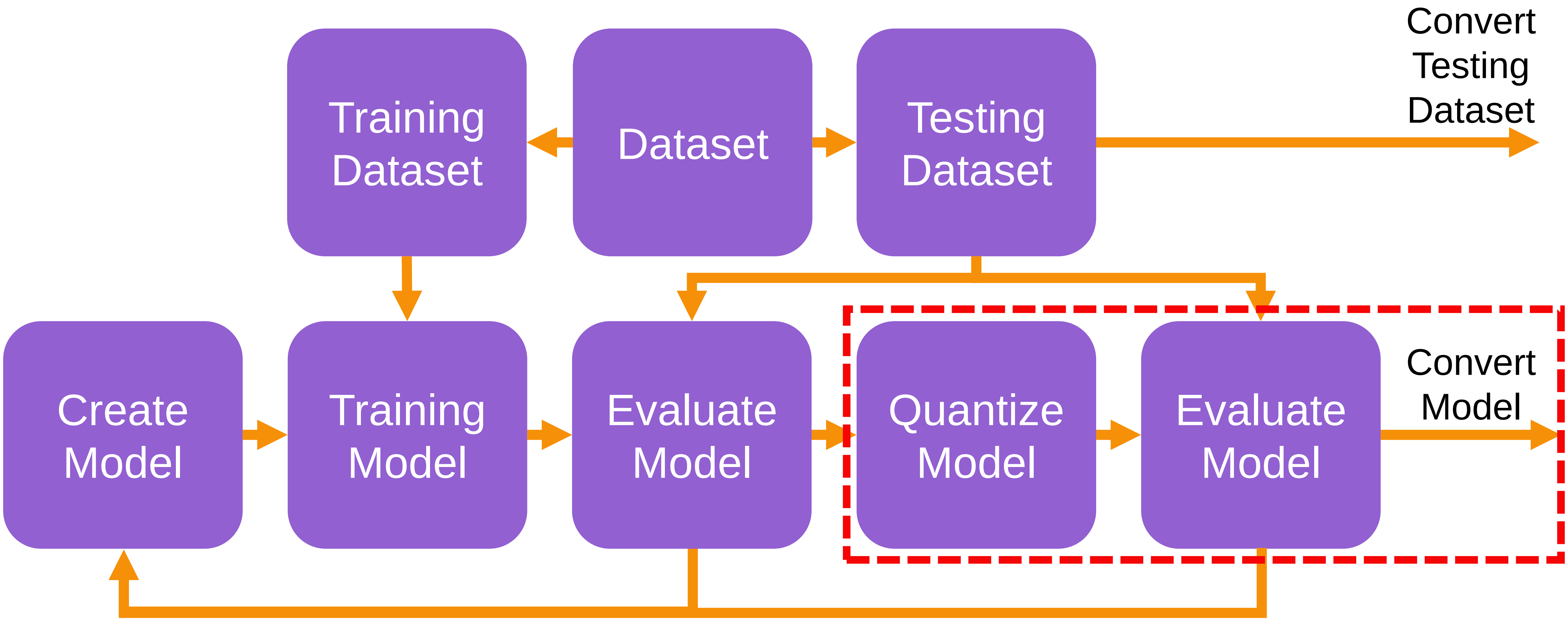
Lab5: EMNIST Letters Recognition (Training)

- Save and load model weights

```
# Save weights of this model
model.save_weights('my_model.h5')
```

```
#Load weights to this TensorFlow model
model.load_weights('my_model.h5')
```

Lab5: EMNIST Letters Recognition (Training)



Lab5: EMNIST Letters Recognition (Training)

- Reload and preprocess images

```
train_images, train_labels = extract_training_samples('letters')
test_images, test_labels = extract_test_samples('letters')

# Make class numbering start at 0
train_labels = train_labels - 1
test_labels = test_labels - 1

preprocessed_test_images = test_images.reshape([test_images.shape[0], img_rows, img_cols, 1])

def thinning(image):
    return np.where(image < 210.0, 0, 255)

preprocessed_test_images = thinning(preprocessed_test_images)
preprocessed_test_images = (preprocessed_test_images - 128.0) / 128.0
```

Lab5: EMNIST Letters Recognition (Training)

- Convert the model to TensorFlow Lite format

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.int8
converter.inference_output_type = tf.int8
```

```
preprocessed_test_images = tf.cast(preprocessed_test_images, tf.float32)
emnist_ds = tf.data.Dataset.from_tensor_slices((preprocessed_test_images)).batch(1)

def representative_data_gen():
    for input_value in emnist_ds.take(100):
        yield [input_value]

converter.representative_dataset = representative_data_gen
```

Lab5: EMNIST Letters Recognition (Training)

- Convert the model to TensorFlow Lite format and save it to a file
- You can convert it to C model. (Later slide)

```
import pathlib

converted_model = converter.convert()

generated_dir = pathlib.Path("generated/")
generated_dir.mkdir(exist_ok=True, parents=True)
converted_model_file = generated_dir/"emnist_model_int8.tflite"
converted_model_file.write_bytes(converted_model)
```

Lab5: EMNIST Letters Recognition (Training)

- Evaluate TensorFlow Lite Model

```
interpreter = tf.lite.Interpreter(model_path=str(converted_model_file))
interpreter.allocate_tensors()

max_samples = 20800
```

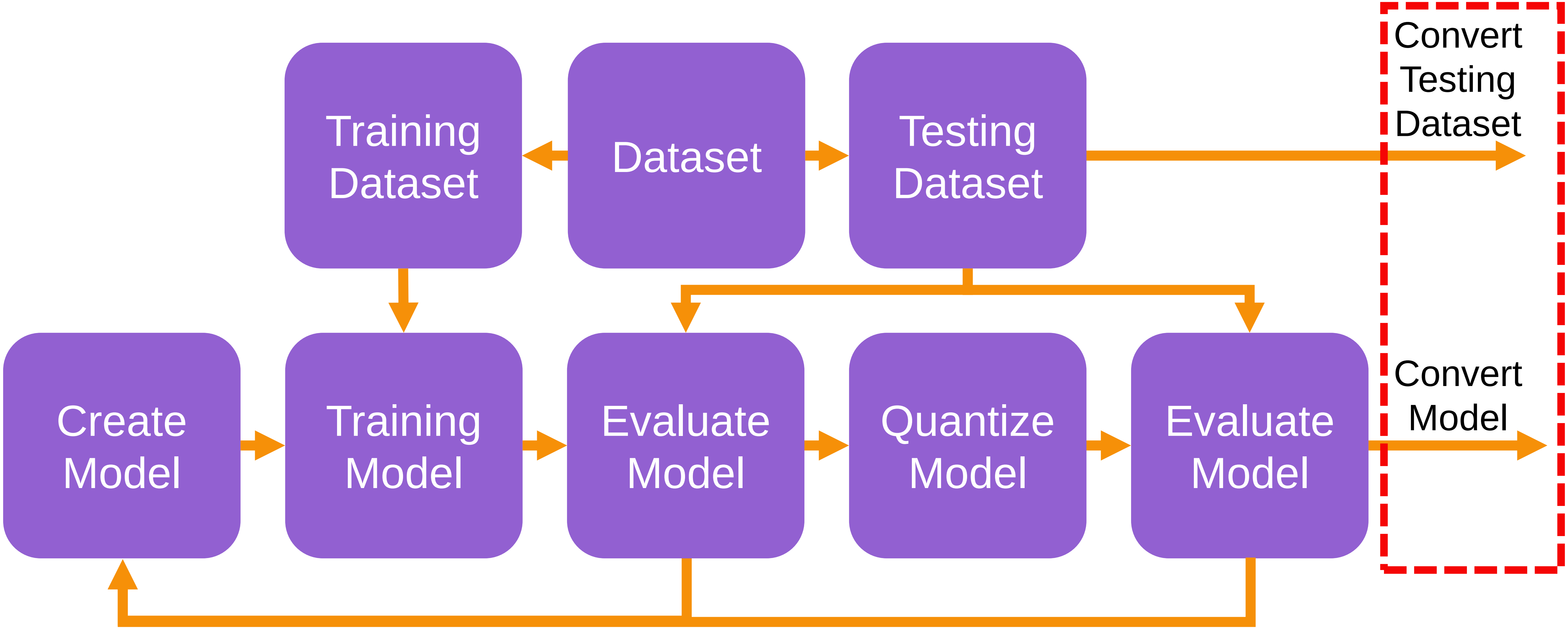
```
# A helper function to evaluate the TF Lite model using "test" dataset.
def evaluate_model(interpreter):
    input_index = interpreter.get_input_details()[0]["index"]
    output_index = interpreter.get_output_details()[0]["index"]
    scale, zero_point = interpreter.get_output_details()[0]['quantization']

    prediction_values = []
```

-

```
print(str(evaluate_model(interpreter)) + "%")
```

```
90.5%
```

Lab5: EMNIST Letters Recognition (Training)



Lab5: EMNIST Letters Recognition (Training)

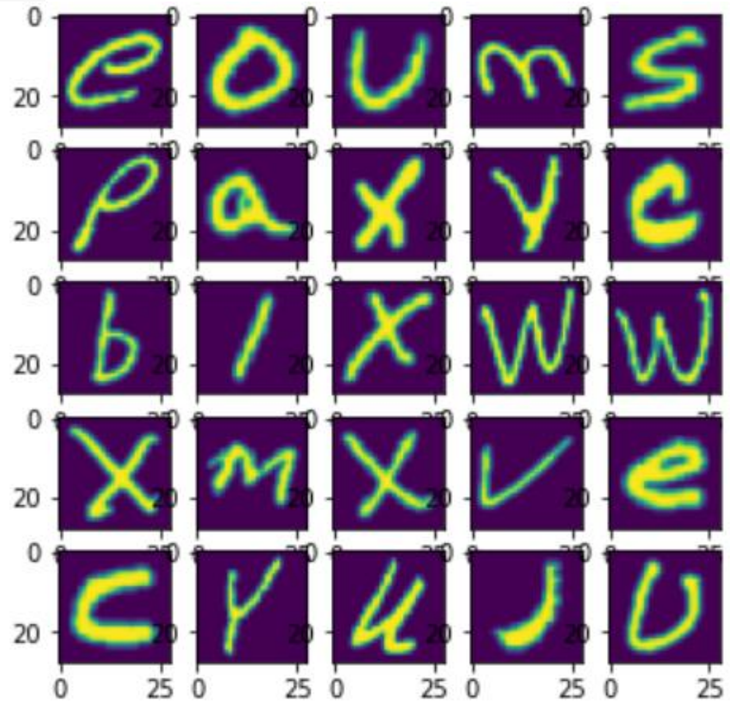
- Convert testing dataset to C file

```
import random

train_images, train_labels = extract_training_samples('letters')
test_images, test_labels = extract_test_samples('letters')

# Make class numbering start at 0
train_labels = train_labels - 1
test_labels = test_labels - 1

num_of_samples = 25
random_test_images = random.sample(range(1, test_images.shape[0]), num_of_samples)
```

Lab5: EMNIST Letters Recognition (Training)

- Convert testing dataset to C file

```
fig=plt.figure(figsize=(5, 5))
cols = 5
rows = 5

for plt_idx, img_idx in enumerate(random_test_images, 1):
    img = test_images[img_idx]
    fig.add_subplot(rows, cols, plt_idx)
    plt.imshow(img)
plt.show()
```



Lab5: EMNIST Letters Recognition (Training)

- Convert testing dataset to C file

```
samples_file = open("generated/test_samples.cc", "w")

samples_file.write("#include \"test_samples.h\"\n\n")
samples_file.write("const int kNumSamples = " + str(num_of_samples) + ";\n\n")

samples = ""
samples_array = "const TestSample test_samples[kNumSamples] = {"

for sample_idx, img_idx in enumerate(random_test_images, 1):
    img_arr = list(np.ndarray.flatten(test_images[img_idx]))
    var_name = "sample" + str(sample_idx)
    samples += "TestSample " + var_name + " = {\n" #+ "[IMAGE_SIZE] = { "
    samples += "\t.label = " + str(test_labels[img_idx]) + ",\n"
    samples += "\t.image = {\n"
    wrapped_arr = [img_arr[i:i + 20] for i in range(0, len(img_arr), 20)]
    for sub_arr in wrapped_arr:
        samples += "\t\t" + str(sub_arr)
    samples += "\t}\n};\n\n"
    samples_array += var_name + ", "
```

Lab5: EMNIST Letters Recognition (Training)

- You will see testing dataset ”generated/test_samples.cc”

```
samples = samples.replace("[", "")
samples = samples.replace("]", ",\n")
samples_array += "};\n"

samples_file.write(samples);
samples_file.write(samples_array);
samples_file.close()
```

Lab5: EMNIST Letters Recognition (Training)

- Open cygwin64 and convert tflite to c file

```
$ cd c:
$ cd Users/{username}/{Workshop_path}
$ cd Lab5_tflm_conversion_training/generated/
$ xxd -i emnist_model_int8.tflite > model.h
```

Lab5: EMNIST Letters Recognition (Training)

- You will see TensorFlow Lite model ”generated/model.h”

```
/cygdrive/c/Users/williet/Documents/VM/himax_tflm/Synopsys_WEI/Example_Pro...

williet@WILLIET-7490 ~
$ cd c:

williet@WILLIET-7490 /cygdrive/c
$ cd Users/williet/Do
Documents/ Downloads/

williet@WILLIET-7490 /cygdrive/c
$ cd Users/williet/Documents/VM/himax_tflm/Synopsys_WEI/Example_Project/

williet@WILLIET-7490 /cygdrive/c/Users/williet/Documents/VM/himax_tflm/Synopsys_
WEI/Example_Project
$ cd Lab5_tflm_emnist_training/generated/

williet@WILLIET-7490 /cygdrive/c/Users/williet/Documents/VM/himax_tflm/Synopsys_
WEI/Example_Project/Lab5_tflm_emnist_training/generated
$ xxd -i emnist_model_int8.tflite > model.h
```

Lab5: EMNIST Letters Recognition (Training)

Conclusion

- Import all the libraries as tools to build a model
- Load dataset and split into training set and testing set
- Preprocess your dataset to a format that your model accept
- Build your model either by API or write by yourself
- Define loss function ,epochs, learning rate …
- Train your model with training set
- Evaluate your model with testing set
- Fine tune your model

Hands-on (Lab 5-2): Integrate TensorFlow Lite and WE-I

Lab5: Integrate TensorFlow Lite and WE-I

- Integrated Project: Lab5_emnist_letter_test

- Copy “Lab5_tflm_emnist_training/generated/model.h” to

  to “ Lab5_emnist_letter_test/inc/model.h“

- Copy “Lab5_tflm_emnist_training/generated/test_samples.cc” to

  to “ Lab5_emnist_letter_test/src/test_samples.cc “

Lab5: Integrate TensorFlow Lite and WE-I

- Set your output labels array in “model_settings.h”

```
constexpr int kNumCols = 28;
constexpr int kNumRows = 28;
constexpr int kNumChannels = 1;

constexpr int kImageSize = kNumCols * kNumRows * kNumChannels;

constexpr int kCategoryCount = 26;
extern const char* kCategoryLabels[kCategoryCount];
```

- In lab5 input width and height are 28*28 and channel is 1 (gray)
- KCategoryCount means output classes

Lab5: Integrate TensorFlow Lite and WE-I

- Set your output labels array in “model_settings.cc”

```
const char* kCategoryLabels[kCategoryCount] = { "A", "B",
            "C", "D", "E", "F", "G", "H", "I", "J",
            "K", "L", "M", "N", "O", "P", "Q", "R",
            "S", "T", "U", "V", "W", "X", "Y", "Z"
};
```

26 Alphabets

Lab5: Integrate TensorFlow Lite and WE-I

- Edit micro_op_resolver for your model (main_function.cc)

  You need to add all layers you used in your model,

  and also count how many different layers to edit <?>.

```
// NOLINTNEXTLINE(runtime-global-variables)
static tflite::MicroMutableOpResolver<6> micro_op_resolver;
micro_op_resolver.AddConv2D();
micro_op_resolver.AddMaxPool2D();
micro_op_resolver.AddFullyConnected();
micro_op_resolver.AddReshape();
micro_op_resolver.AddSoftmax();
micro_op_resolver.AddRelu();        Find related code in Synopsys WE
```

```
model.add(Conv2D(filters=16,
                 kernel_size=(filter_x, filter_y),
                 padding="same",
                 input_shape=input_shape))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(MaxPooling2D())
```

```
model.add(Flatten())
model.add(Dense(64))
model.add(BatchNormalization())
model.add(Activation("relu"))

#FC2
model.add(Dense(num_classes))
model.add(Activation("softmax"))
```

Lab5: Integrate TensorFlow Lite and WE-I

- Edit micro_op_resolver for your model (main_function.cc)

```
// NOLINTNEXTLINE(runtime-global-variables)
static tflite::MicroMutableOpResolver<6> micro_op_resolver;
micro_op_resolver.AddConv2D();
micro_op_resolver.AddMaxPool2D();
micro_op_resolver.AddFullyConnected();
micro_op_resolver.AddReshape();
micro_op_resolver.AddSoftmax();
micro_op_resolver.AddRelu();        Find related code in Synopsy
```

```
model.add(Conv2D(filters=16,
                 kernel_size=(filter_x, filter_y),
                 padding="same",
                 input_shape=input_shape))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(MaxPooling2D())
```

```
model.add(Flatten())
model.add(Dense(64))
model.add(BatchNormalization())
model.add(Activation("relu"))

#FC2
model.add(Dense(num_classes))
model.add(Activation("softmax"))
```

Lab5: Integrate TensorFlow Lite and WE-I

- You can find useable functions in “micro_mutable_op_resolver.h”

```
TfLiteStatus AddConv2D() {        Find related code in himax_tflm
  return AddBuiltin(BuiltinOperator_CONV_2D,
                    tflite::ops::micro::Register_CONV_2D(), ParseConv2D);
}

TfLiteStatus AddCos() {
  return AddBuiltin(BuiltinOperator_COS, tflite::ops::micro::Register_COS(),
                    ParseCos);
}

TfLiteStatus AddDepthwiseConv2D() {
  return AddBuiltin(BuiltinOperator_DEPTHWISE_CONV_2D,
                    tflite::ops::micro::Register_DEPTHWISE_CONV_2D(),
                    ParseDepthwiseConv2D);
}
```

91   OUTPUT   DEBUG CONSOLE   TERMINAL

Lab5: Integrate TensorFlow Lite and WE-I

- Declare TensorFlowlite input and output buffer in “main_function.cc”

```
TfLiteTensor* input = nullptr;
TfLiteTensor* output = nullptr;
```

Declare in/ouput pointer to TfliteTensor type variables

```
// Get information about the memory area to use for the model's input.
input = interpreter->input(0);
output = interpreter->output(0);
```

Point it to model in/out

Lab5: Integrate TensorFlow Lite and WE-I

- Make sure your Tesnor area size is large enough in “main_function.cc”

```
// An area of memory to use for input, output, and intermediate arrays.
constexpr int kTensorArenaSize = 136 * 1024;
static uint8_t tensor_arena[kTensorArenaSize];
}  // namespace
```

Lab5: Integrate TensorFlow Lite and WE-I

- Read test_samples data and normalize to either -128 or 127

```
for (int j = 0; j < kNumSamples; j++)
{

  TF_LITE_REPORT_ERROR(error_reporter, "Test sample[%d] Start Reading and round values to either -128 or 127\n", j);
  // Perform image thinning (round values to either -128 or 127)
  // Write image to input data
  for (int i = 0; i < kImageSize; i++) {
    input->data.int8[i] = (test_samples[j].image[i] <= 210) ? -128 : 127;
  }
```

- Start invoking (running model)

```
TF_LITE_REPORT_ERROR(error_reporter, "Test sample[%d] Start Invoking\n", j);
// Run the model on this input and make sure it succeeds.
if (kTfLiteOk != interpreter->Invoke()) {
  TF_LITE_REPORT_ERROR(error_reporter, "Invoke failed.");
}
```

Lab5: Integrate TensorFlow Lite and WE-I

- Record output

```
TF_LITE_REPORT_ERROR(error_reporter, "Test sample[%d] Start Finding Max Value\n", j);
// Get max result from output array and calculate confidence
int8_t* results_ptr = output->data.int8;
int result = std::distance(results_ptr, std::max_element(results_ptr, results_ptr + 26));
float confidence = ((results_ptr[result] - zero_point)*scale + 1) / 2;
const char *status = result == test_samples[j].label ? "SUCCESS" : "FAIL";
```

- Declare a int8_t ptr to output data
- Find max element from output[0] to output[25]
- Calculate probability
- Status to record “SUCCESS” or “FAIL”

Lab5: Integrate TensorFlow Lite and WE-I

- Print Test_sample numbers and output result in Cygwin64 terminal

```
TF_LITE_REPORT_ERROR(error_reporter,
  "Test sample \"%s\":\n"
  "Predicted %s (%d%%) - %s\n",
  kCategoryLabels[test_samples[j].label],
  kCategoryLabels[result], (int)(confidence * 100), status);
```

- Output will be like this picture

```
Test sample[5] Start Reading and round values to either -128 or 127

Test sample[5] Start Invoking

Test sample[5] Start Finding Max Value

Test sample "T":
Predicted K (66%) - FAIL

Test sample[5] Start Reading

Test sample[5] Start Invoking

Test sample[5] Start Finding Max Value

Test sample "T":
Predicted T (93%) - SUCCESS

Test sample[5] Finsih


Test sample[6] Start Reading and round values to either -128 or 127
```

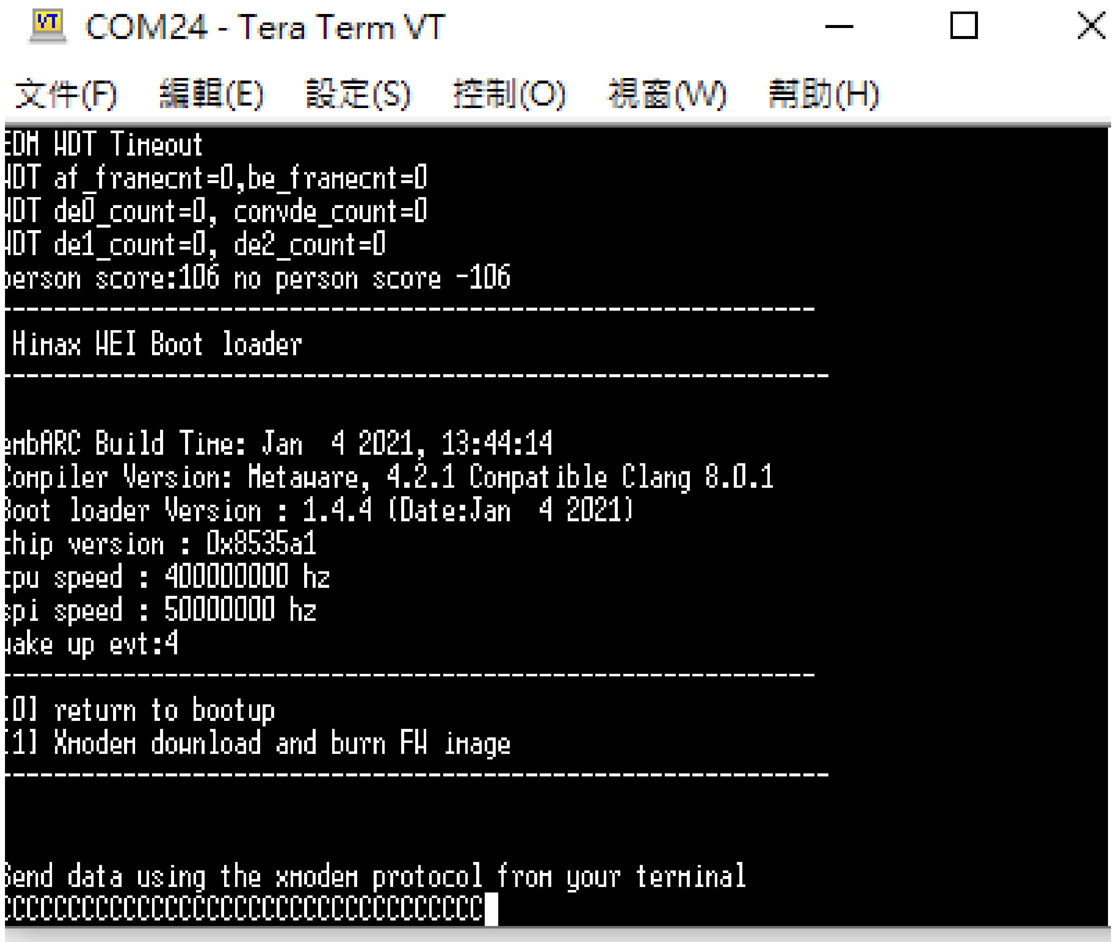
Lab5: Integrate TensorFlow Lite and WE-I

Conclusion

- Convert TensorFlow Lite model to “model.h”
- Copy “model.h” to “Your_Project/inc”
- (Option) Convert TensorFlow Lite testing dataset to “test_samples.cc”
- (Option) Copy “test_samples.cc” to “Your_Project/src”
- Set your model setting in “model settings.cc” and “model settings.h”
- Edit micro_op_resolver for your model in “main_function.cc”
- Edit TensorFlow input and output buffer in “main_function.cc”
- Develop your project

Lab5: Integrate TensorFlow Lite and WE-I

- Make Flash, push WE-I reset button, and download image file to WE-I



Lab5: Integrate TensorFlow Lite and WE-I

- This example project will test random “A” to “Z” for recognizing
- Accuracy will show at the end
- You can record log file by log function