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

Willie Tsai

2023/02/06

ARC DESIGN CONTEST 2021

SYNOPSYS®
Silicon to Software™

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

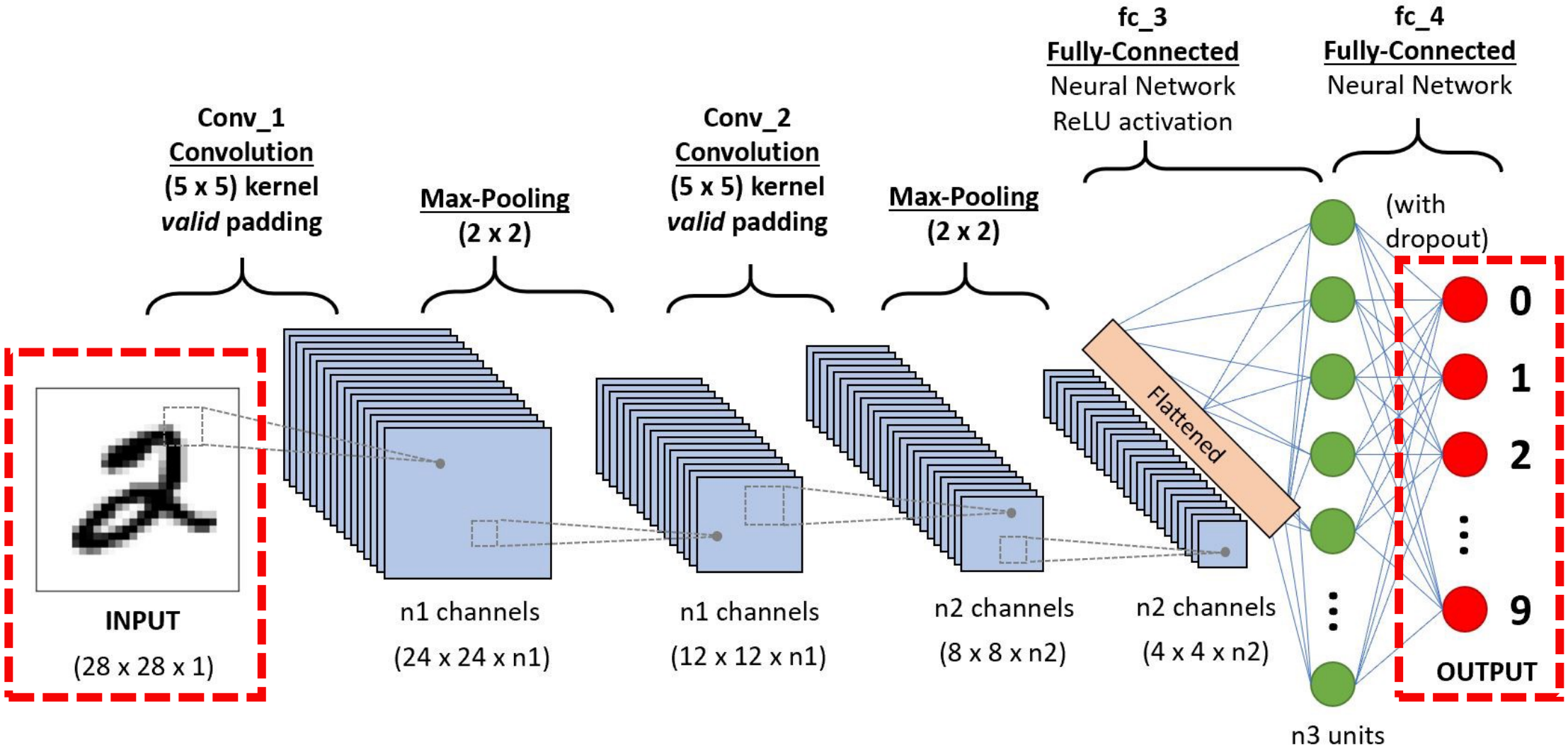
Output (Prediction)

0.2	Dog
0.7	Cat
0.1	Bird

(Ground truth)

0
1
0

CNN - Architecture

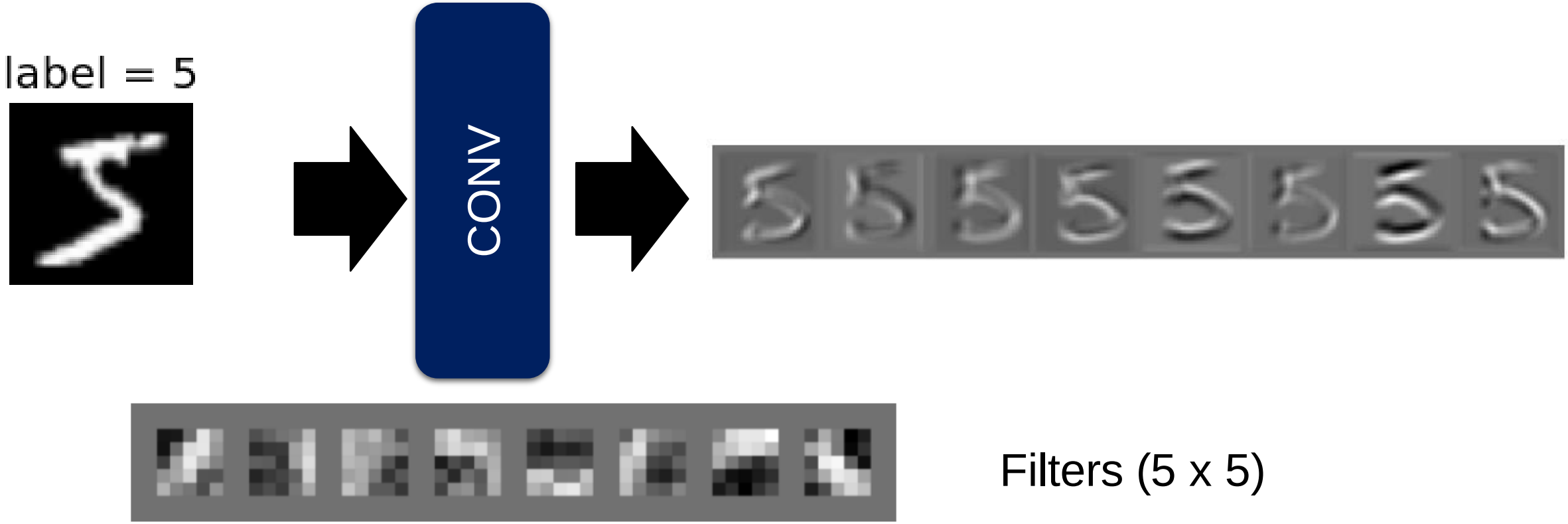


CNN - Components

- Convolution
- Activation Function
- Pooling
- Fully Connected

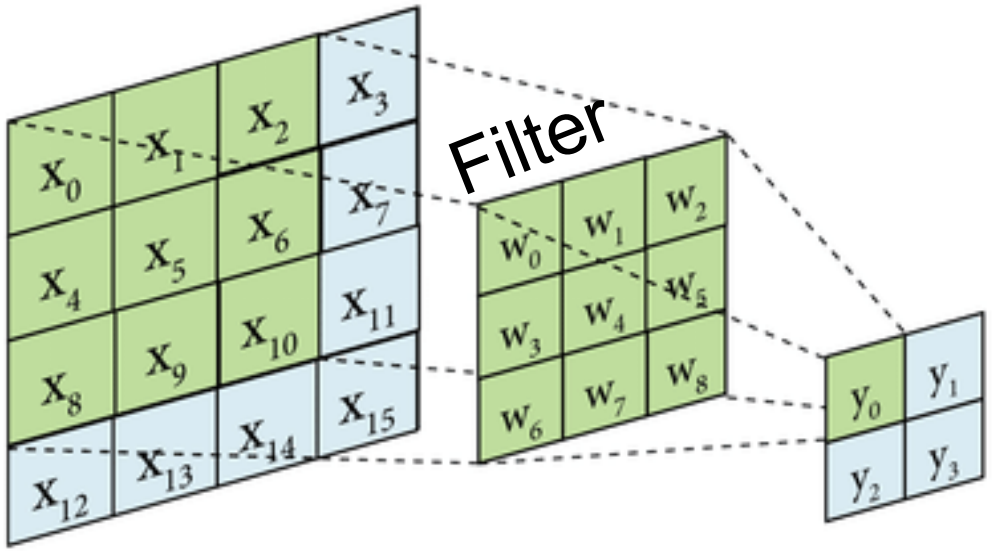
Convolution

- Filters are used for extracting features in an image.



Convolution

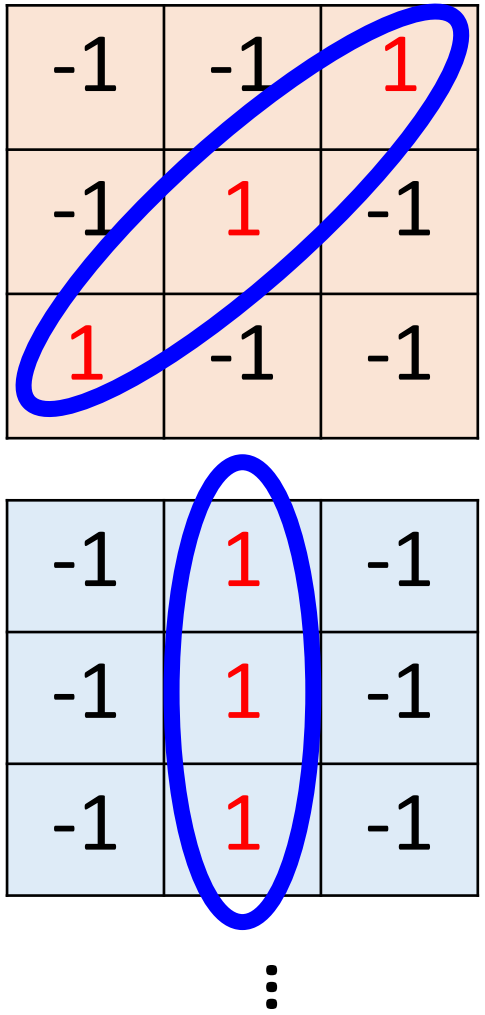
- It operates as a sliding window that sweep across the whole image detecting features.



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



3x3 Filter 1

3x3 Filter 2

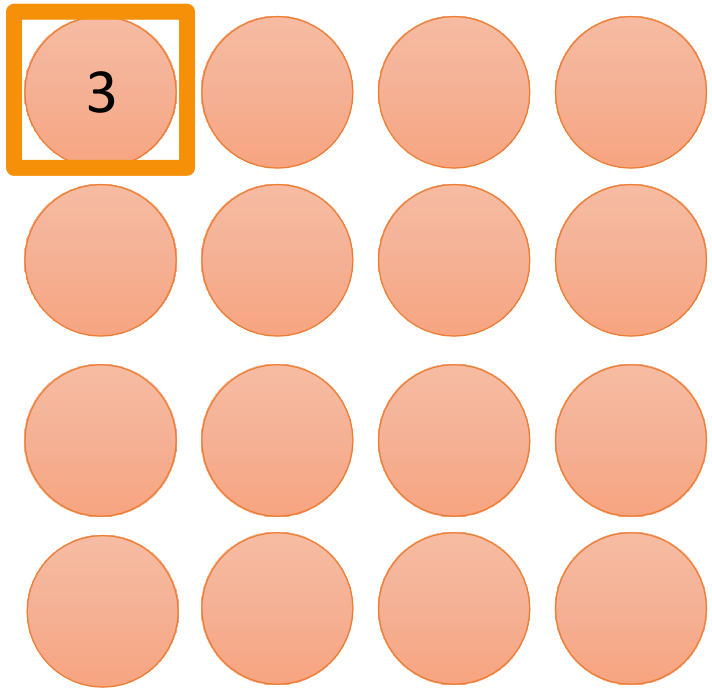
Convolution

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

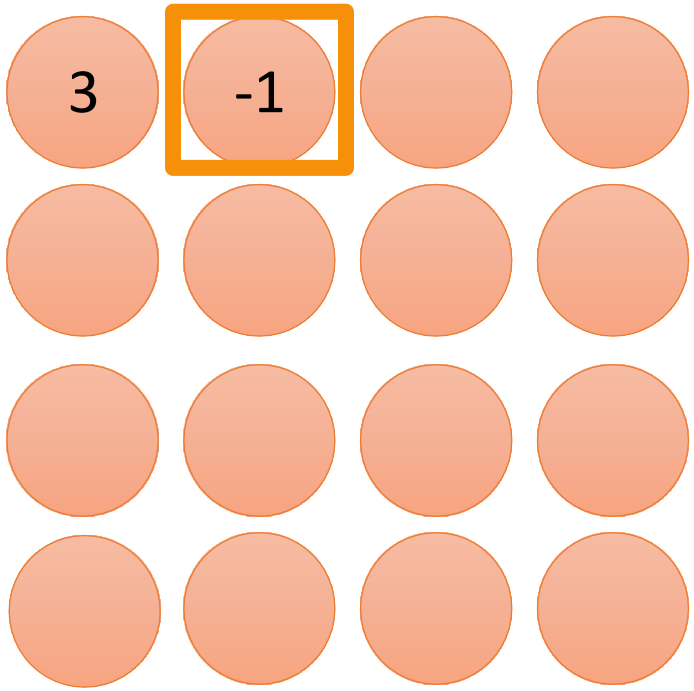
Convolution

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

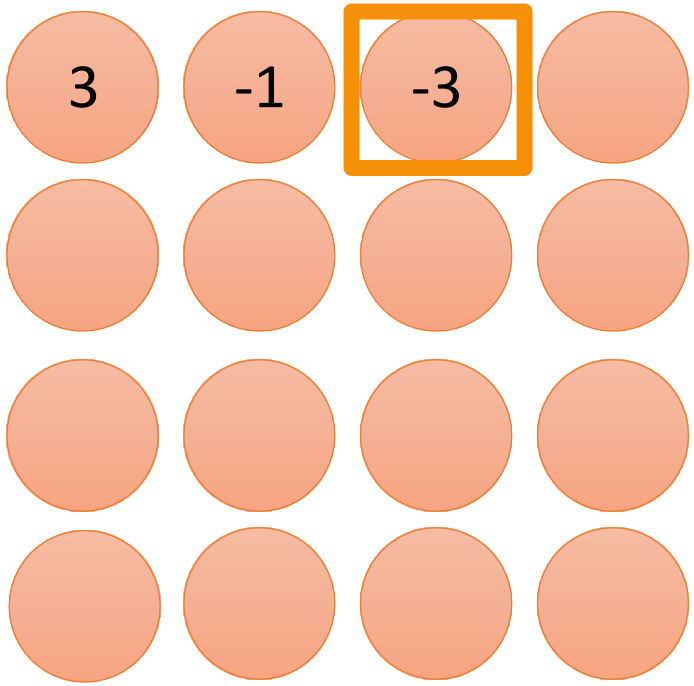
Convolution

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



Feature Map

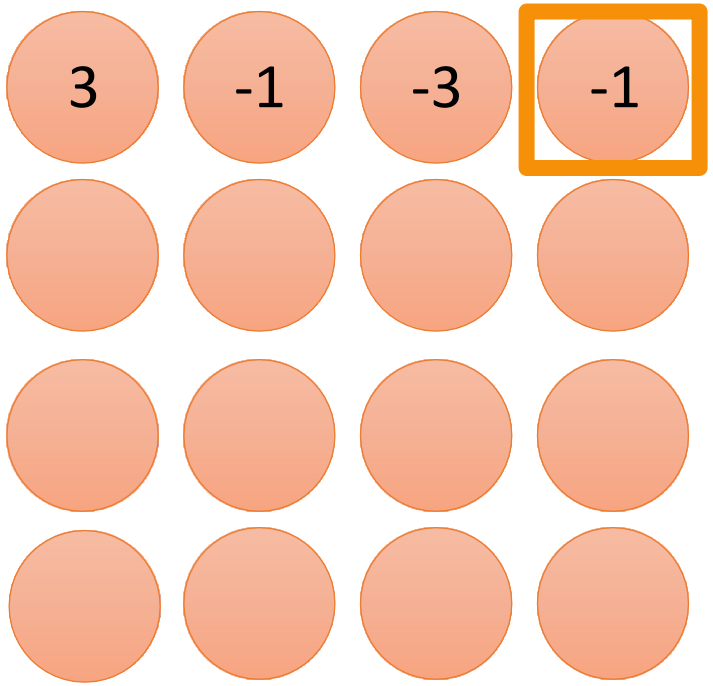
Convolution

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

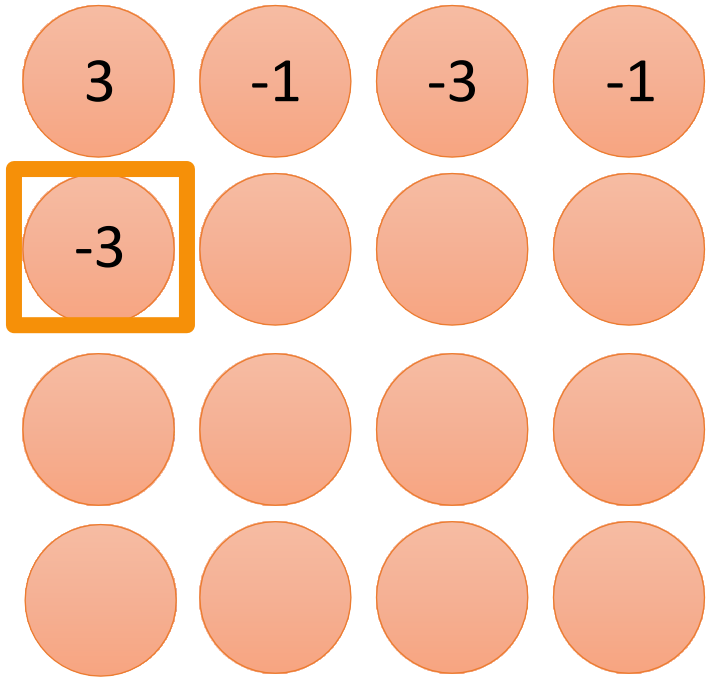
Convolution

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



Feature Map

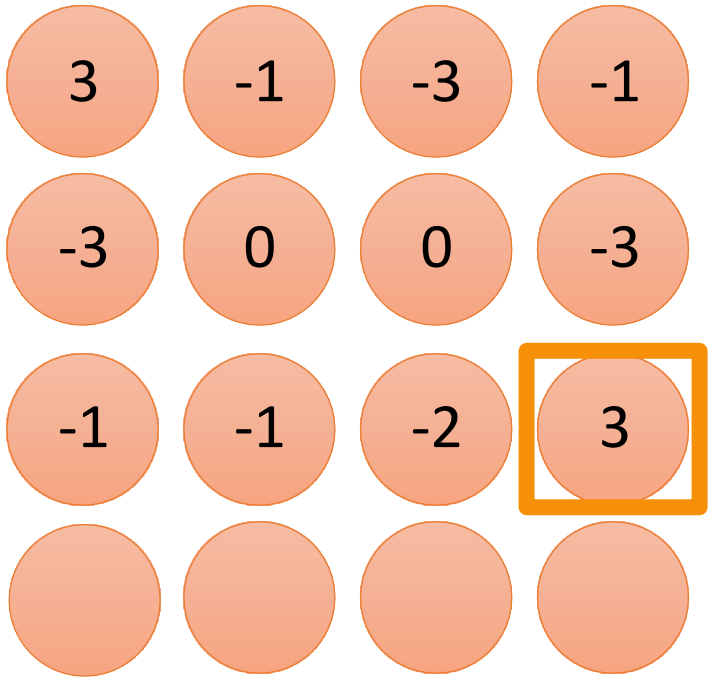
Convolution

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	-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

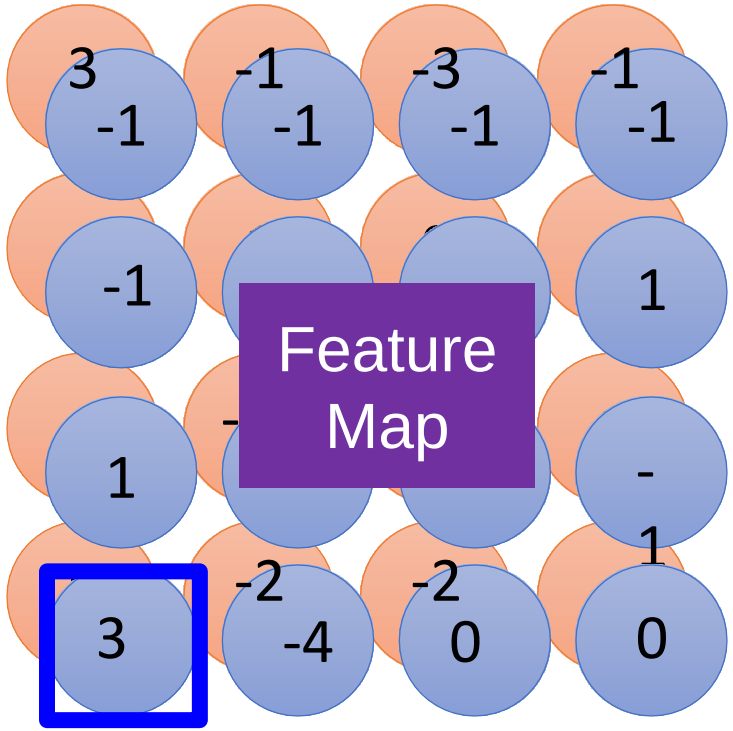
Convolution

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

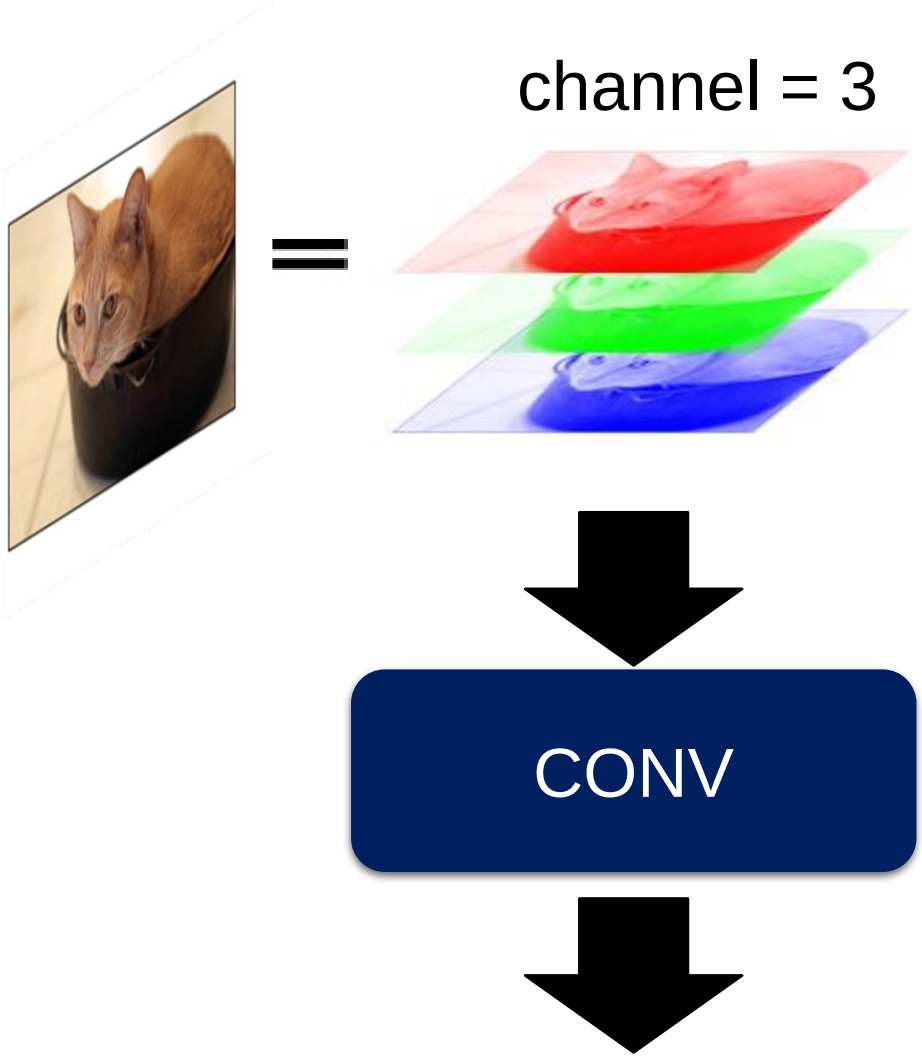
6 x 6 binary image

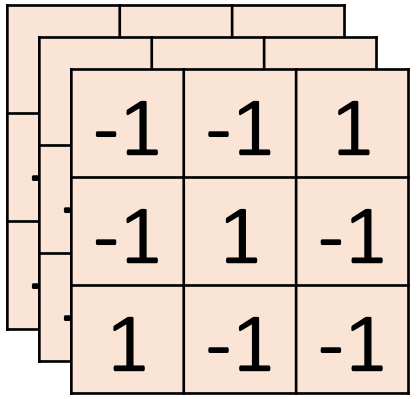
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

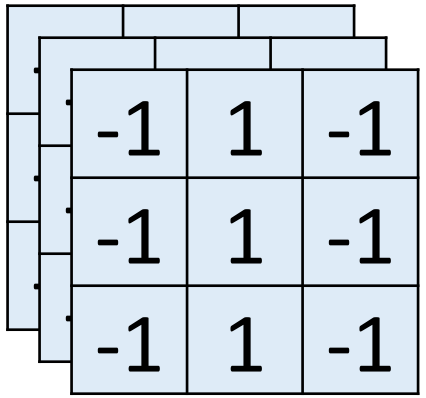


Convolution





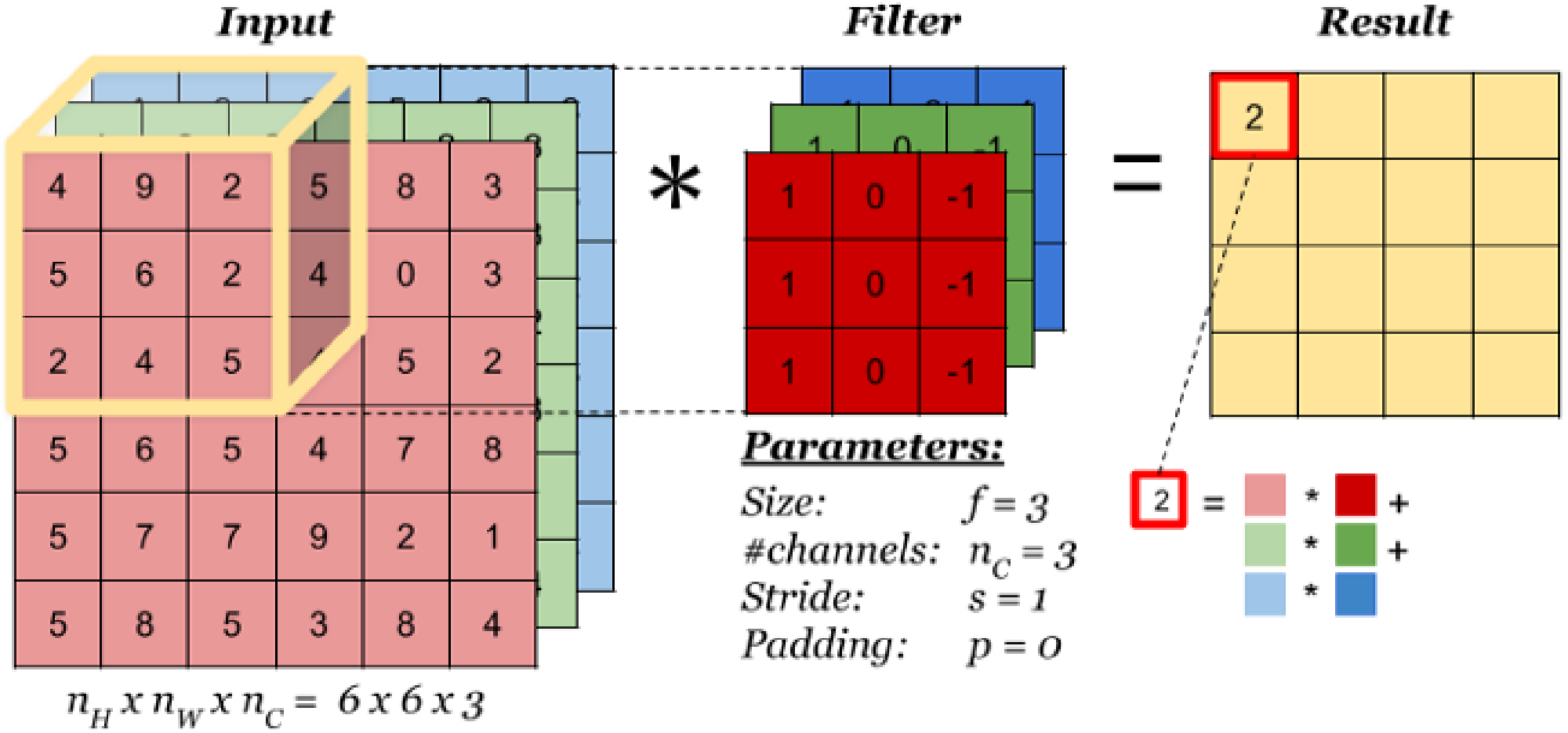
Filter 1
3 x 3 x #channel



Filter 2
3 x 3 x #channel

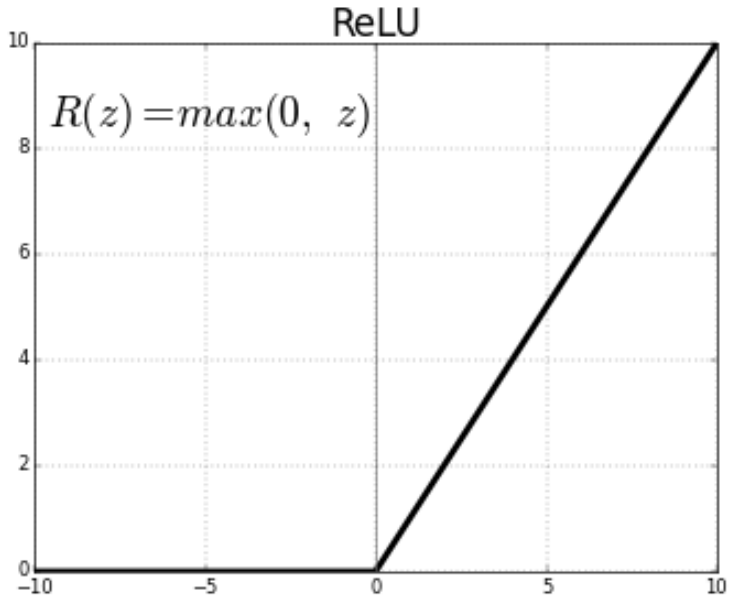
Convolution

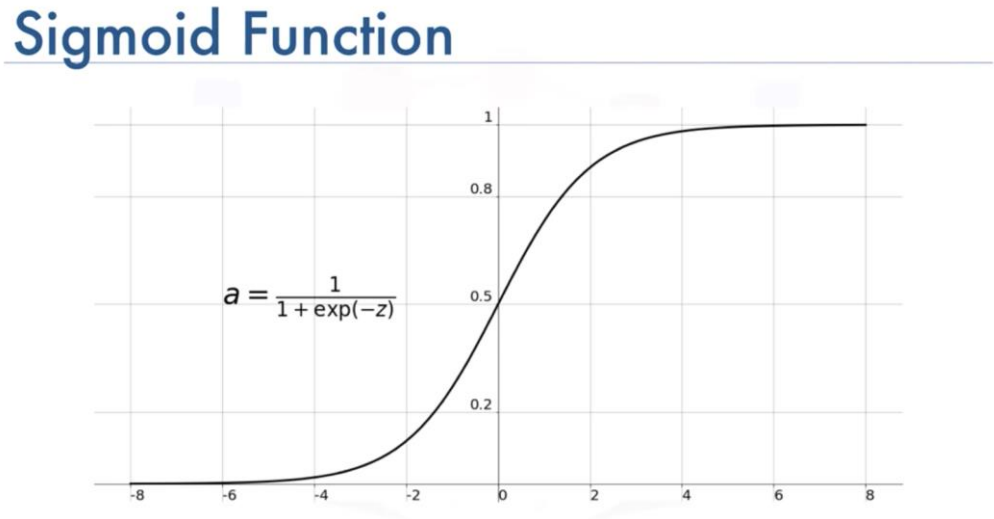
- Convolution is the first layer to extract features from an input image



Activation Function

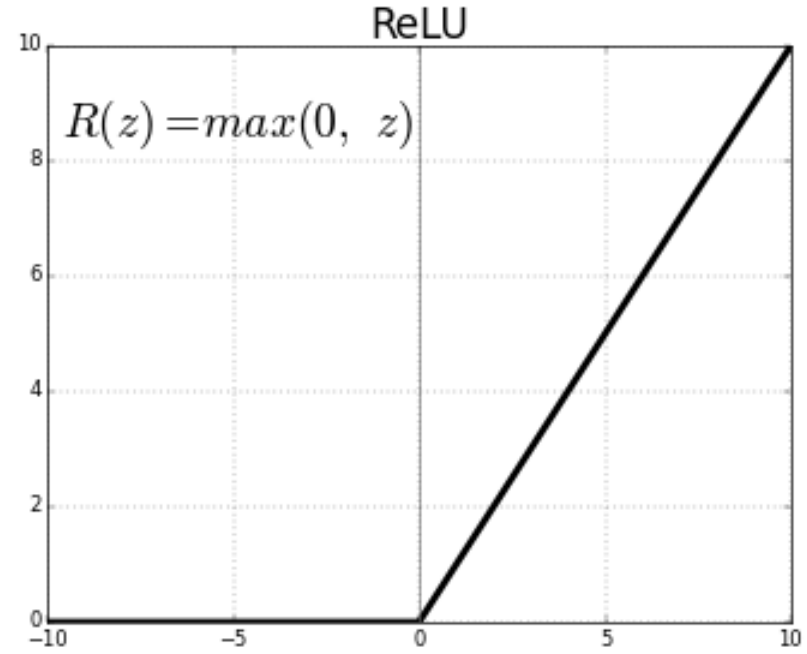
- Activation functions determine the output of current layer (input of the next layer)
- Commonly used activation functions are ReLU, Tanh, Sigmoid and Softmax.





Example – ReLU

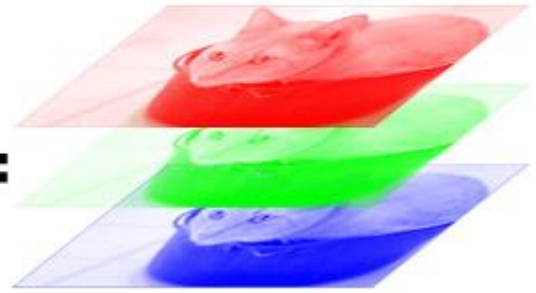
15	20	**-10**	35
18	**-11**	25	99
20	**-15**	25	**-10**
11	75	18	23



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

CONV + ReLU

=



CONV

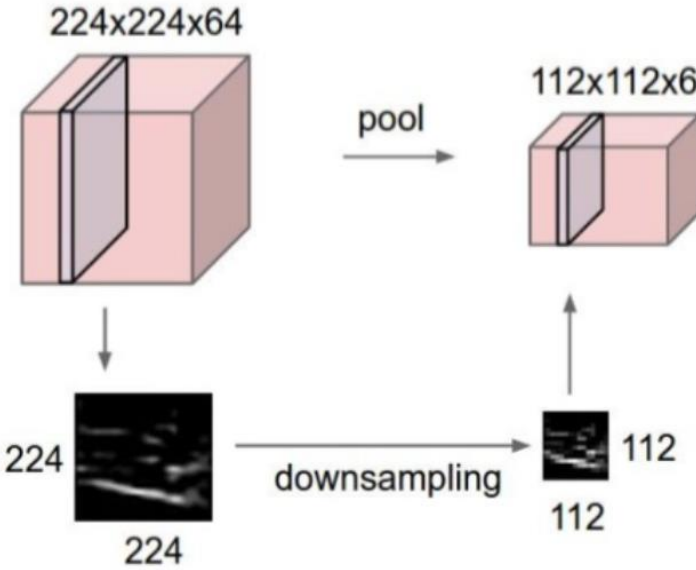
ReLU

-1	1	-1	7
5	-2	-2	1
1	-1	12	-9
-1	0	-4	3

0	1	0	7
5	0	0	1
1	0	12	0
0	0	0	3

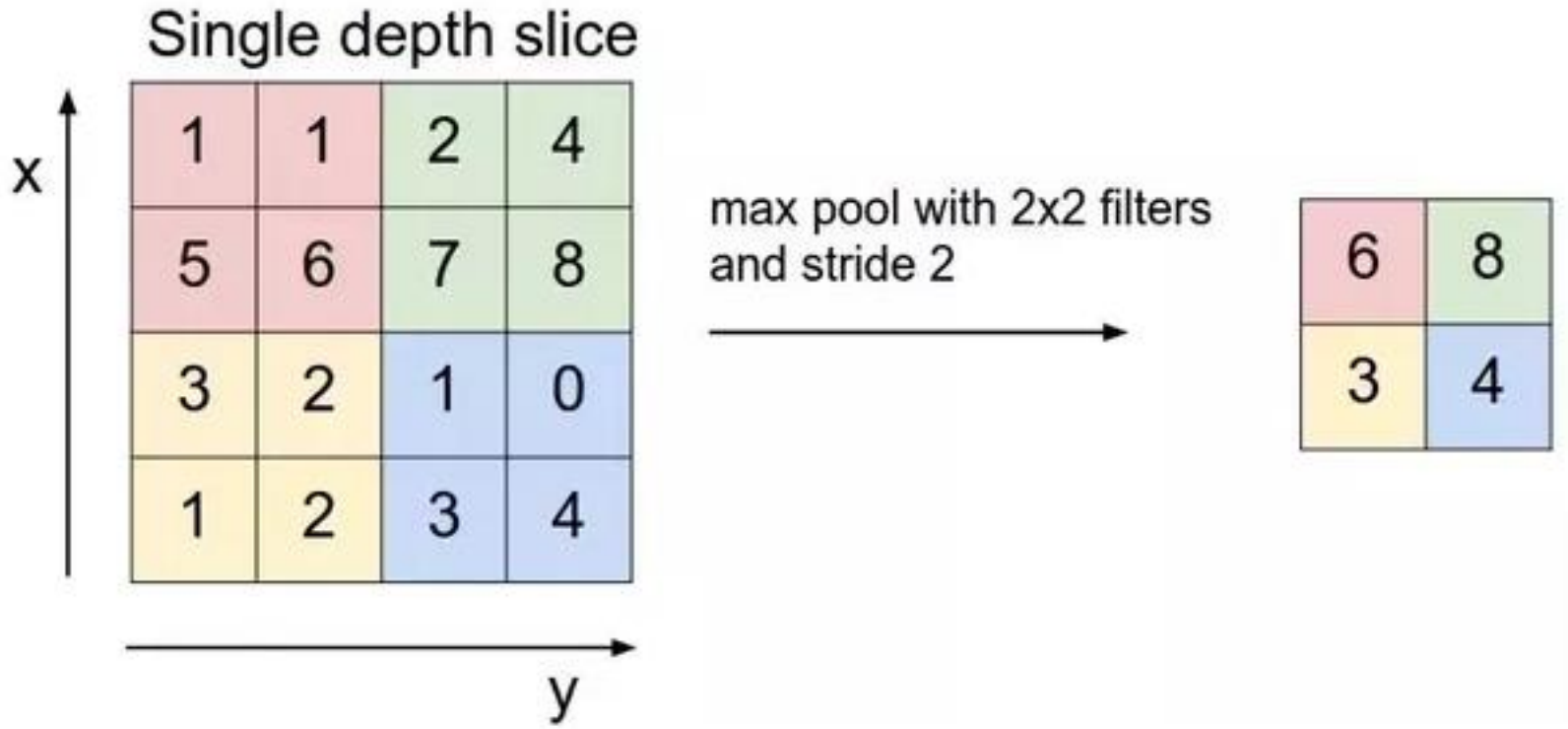
Pooling

- 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.

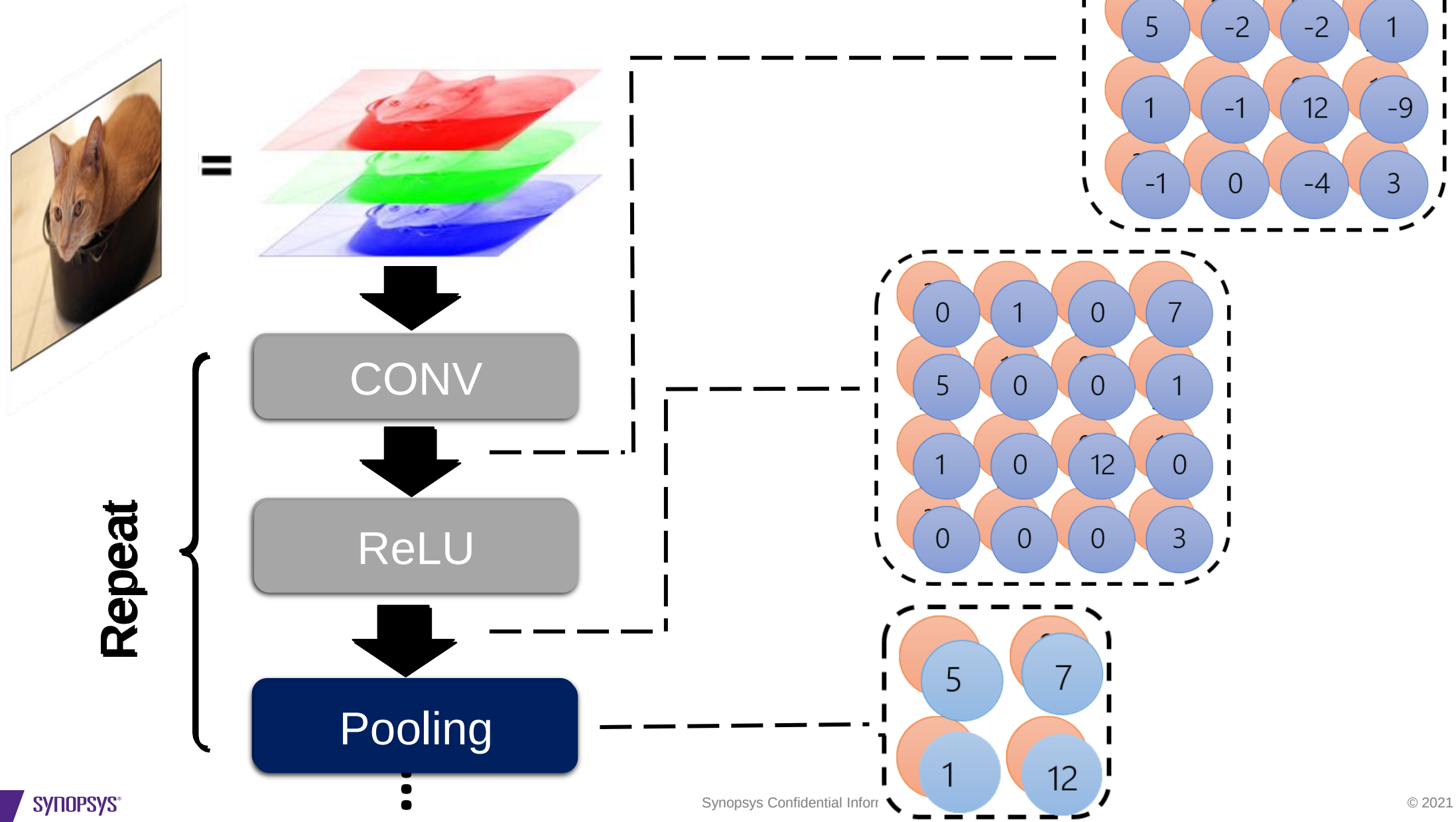


Example – Max pooling

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

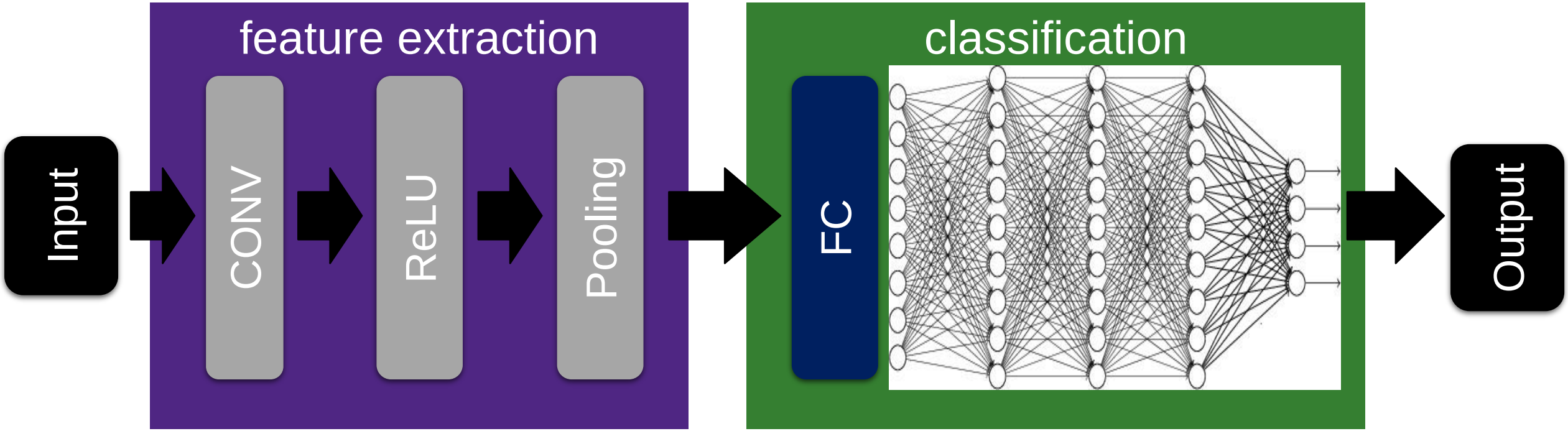


CONV + ReLU + Pooling

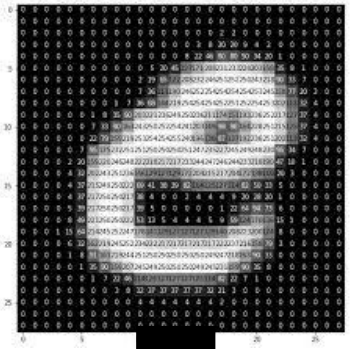


Fully Connected

- 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



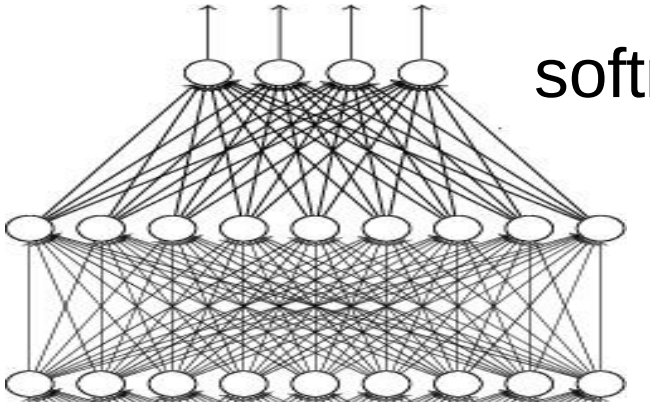
CONV

BatchNorm

ReLU

Max pooling

Flatten



softmax

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

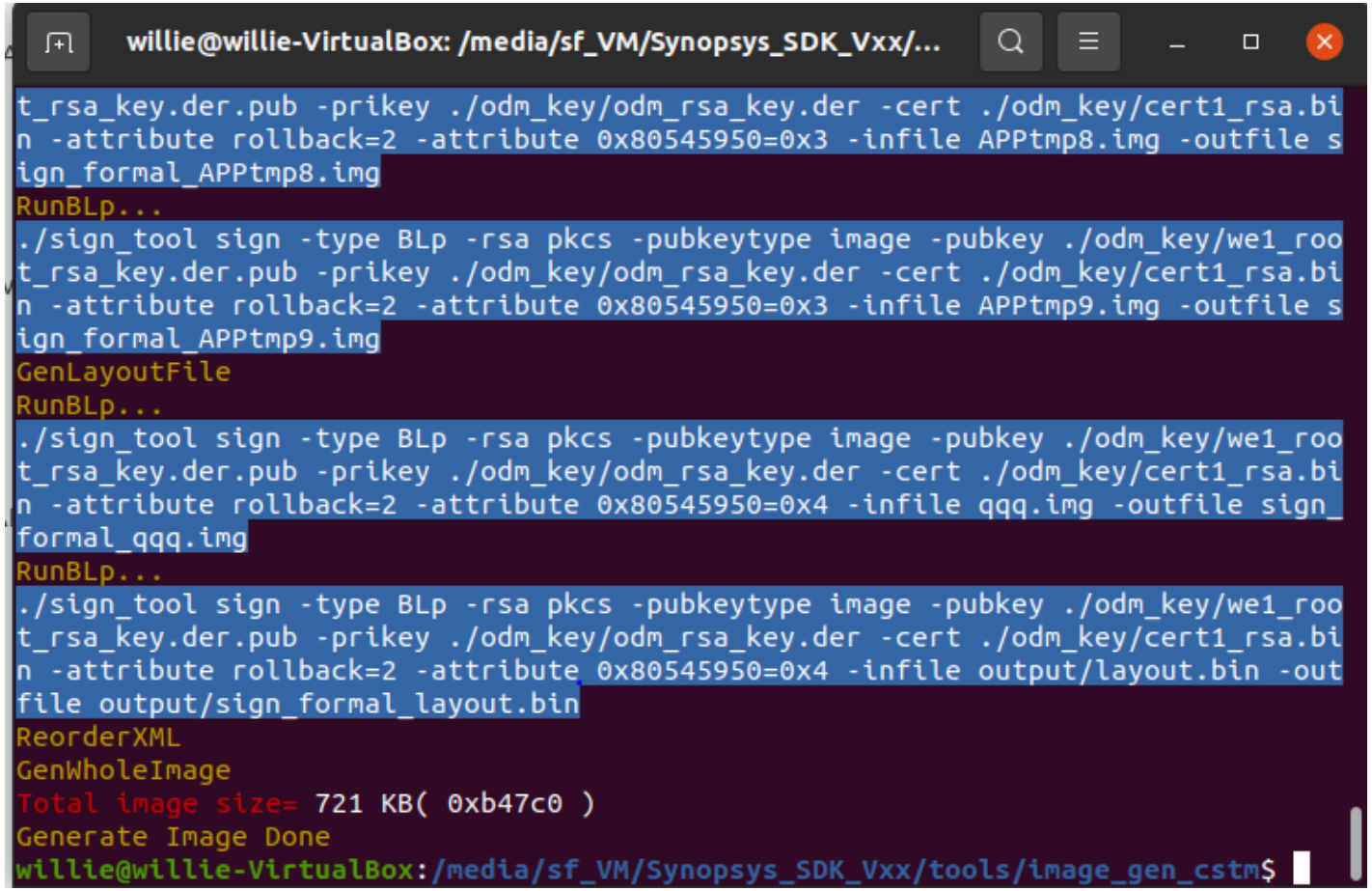
Reference

- Hung-yi Lee, 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

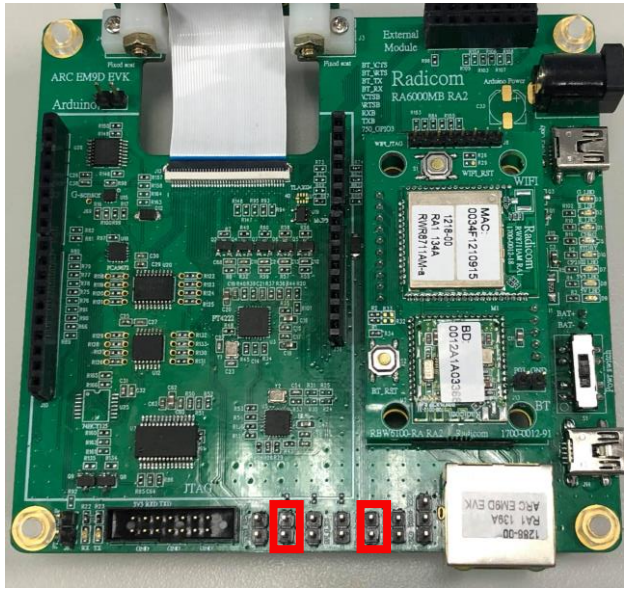
Lab4: Person Detection

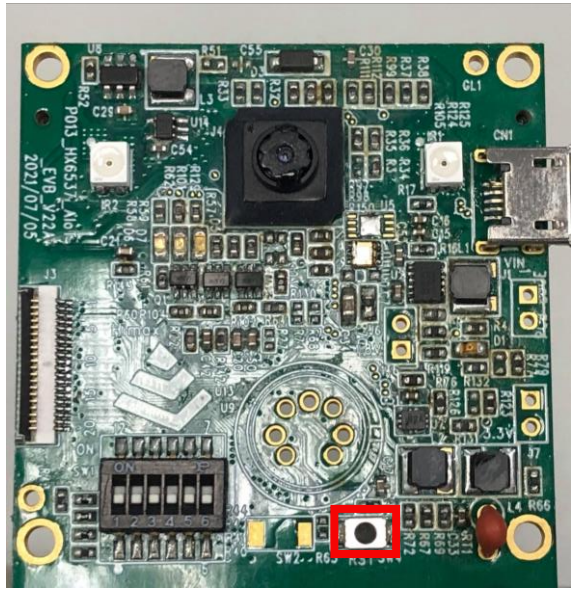
- Make project “Lab4_tflm_person_detect” and convert to the image file



Lab4: Person Detection

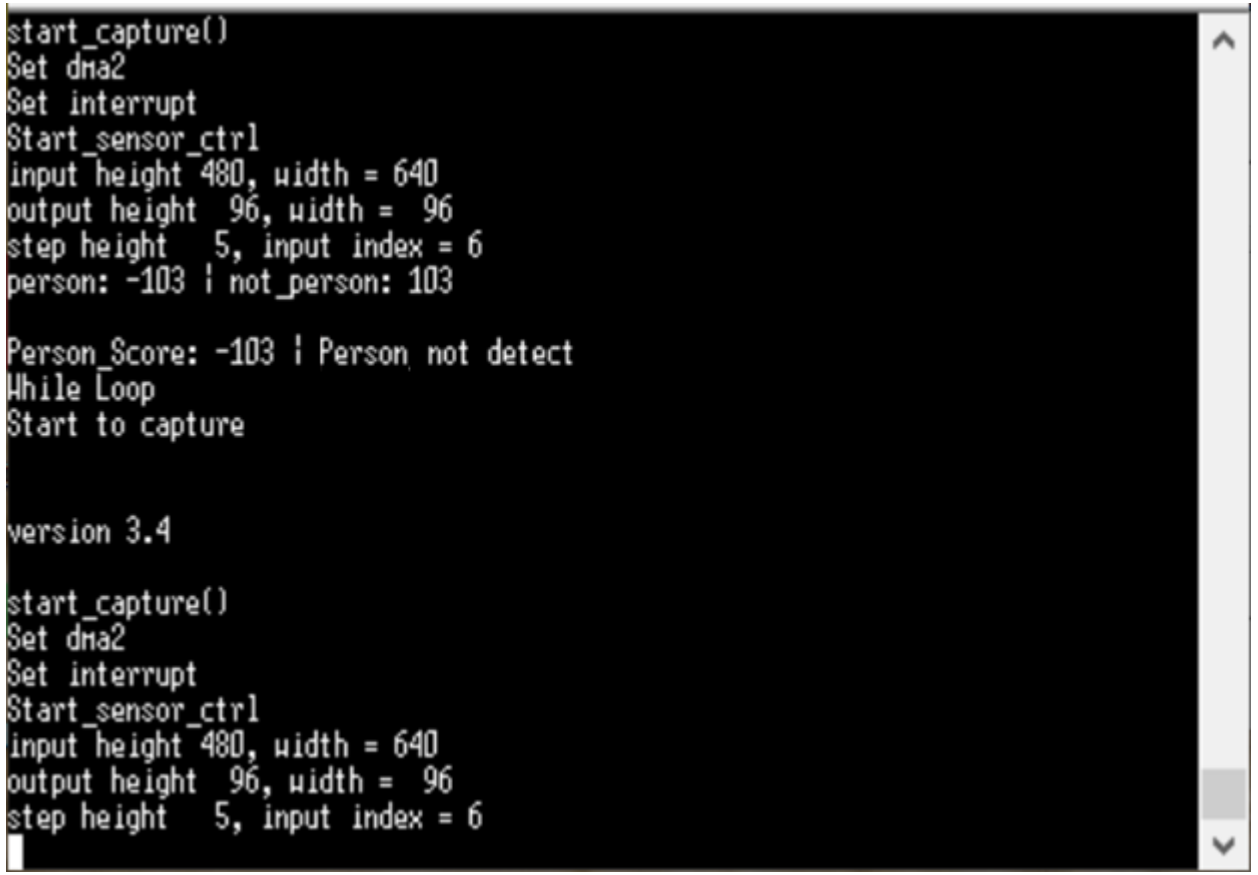
1. Short J20 and J11 for update application mode
2. Download image file to CPU
3. Open J20 for run mode
4. Press reset button SW4. MCU will reset and run the application





Lab4: Person Detection

- This example project will detect person by camera.
- If person score > 0, it will show “Person detect”

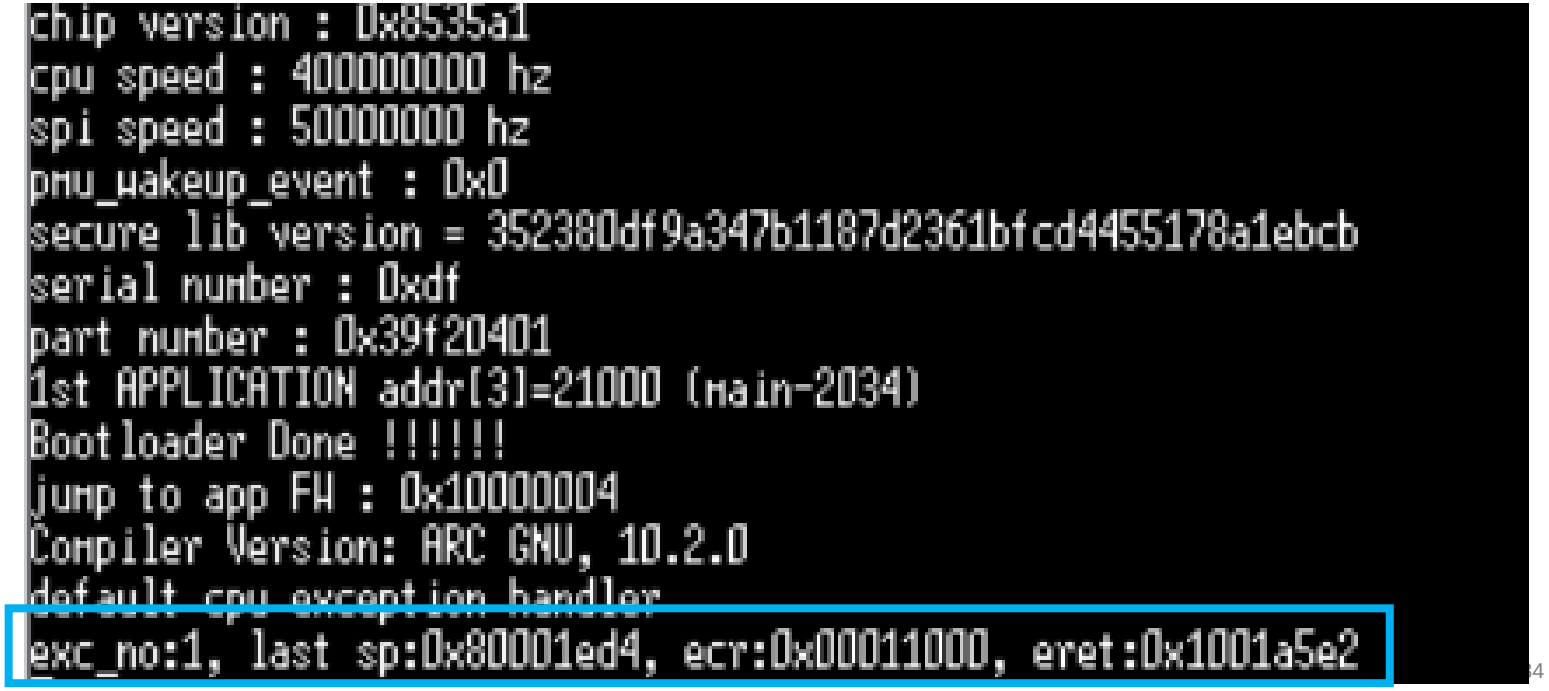


Lab4: Person Detection

- If you get this error message after program and reset CPU.

  You can try to turn off all power and turn on again.
- If it still shows this error message, please program your flash again!!!
- Or you can edit your project, compile and convert a new image file,

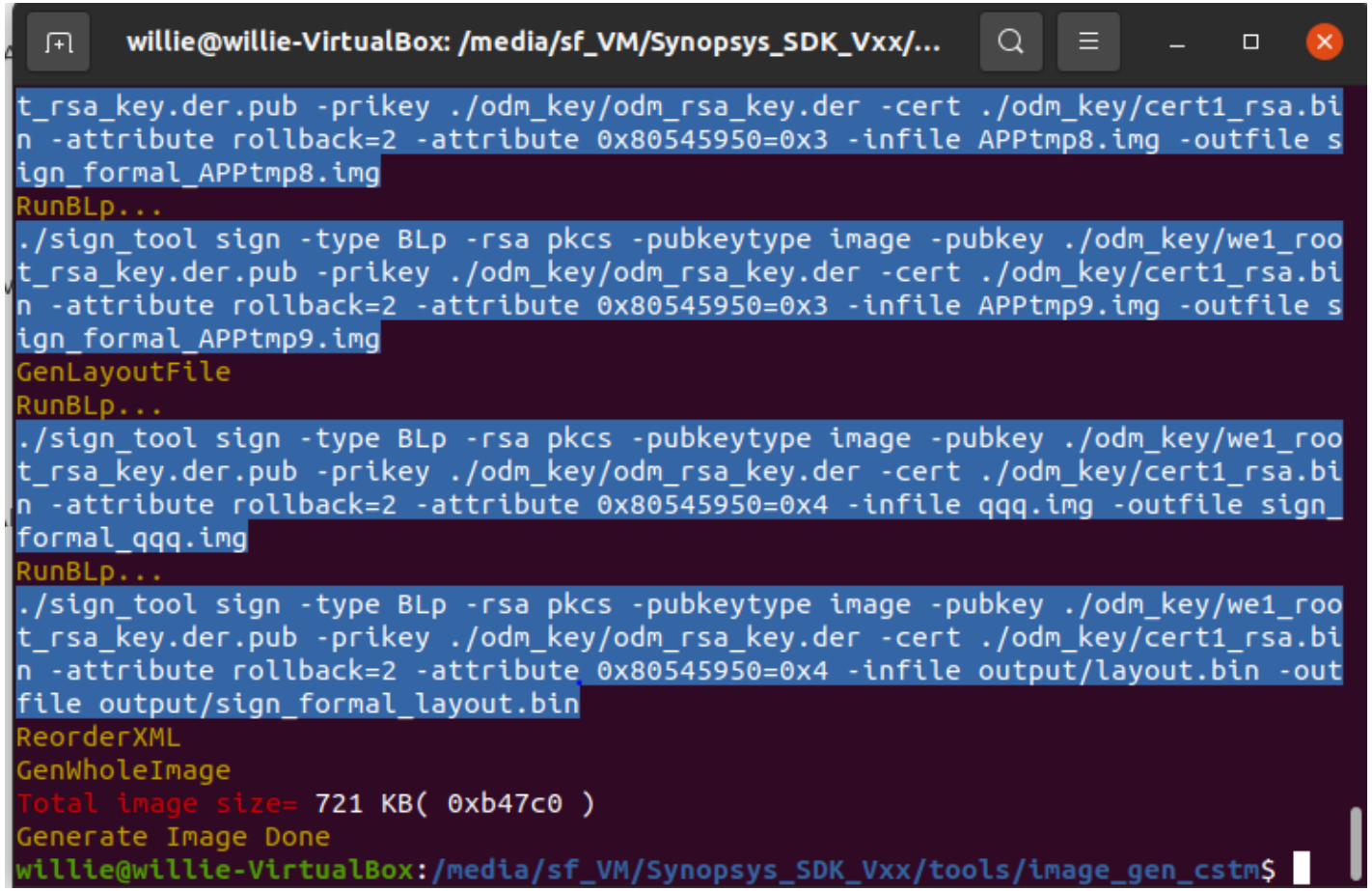
  then program it again.



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

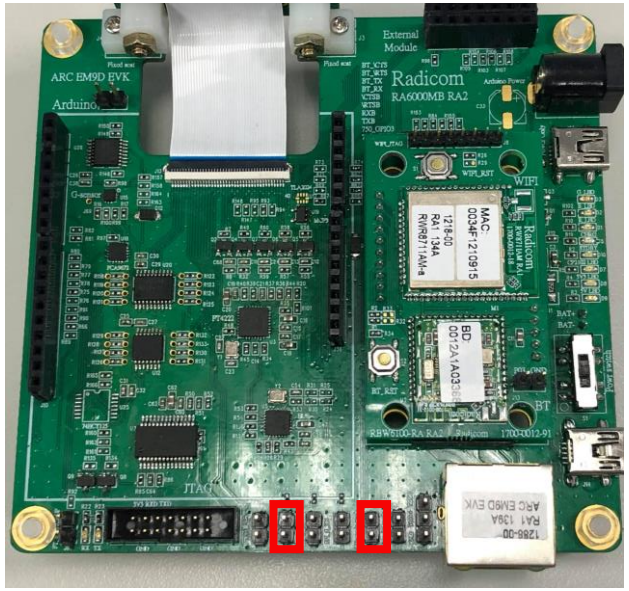
Lab4: Hand-Writing Number Recognition

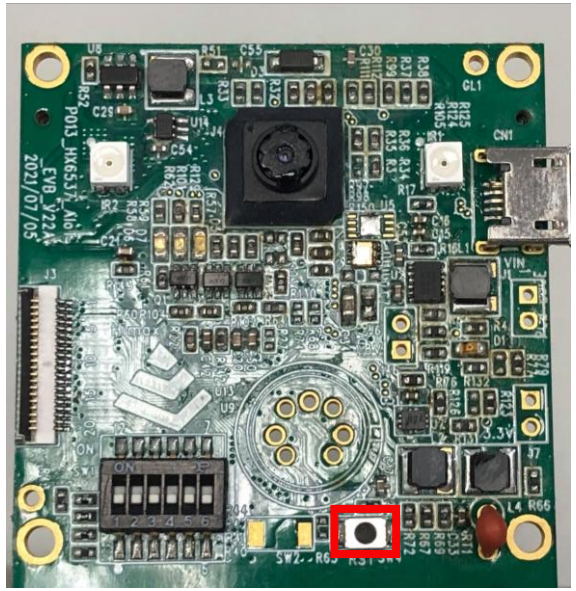
- Make project “Lab4_tflm_emnist_number” and convert to the image file



Lab4: Hand-Writing Number Recognition

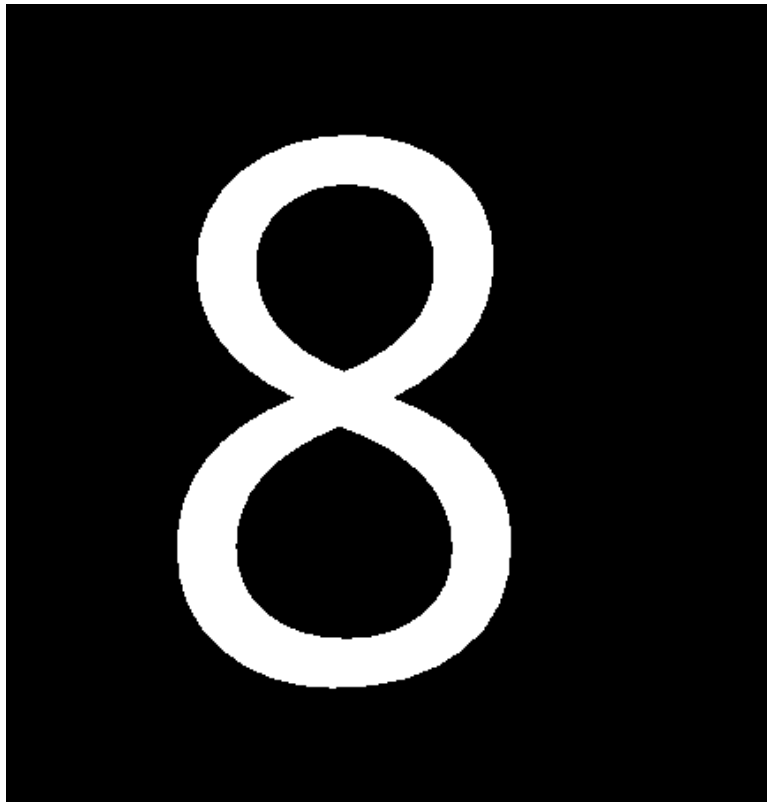
1. Short J20 and J11 for update application mode
2. Download image file to CPU
3. Open J20 for run mode
4. Press reset button SW4. MCU will reset and run the application





Lab4: Hand-Writing Number Recognition

- This example project will recognize number by camera
- Please make sure the number is in the middle of the image.



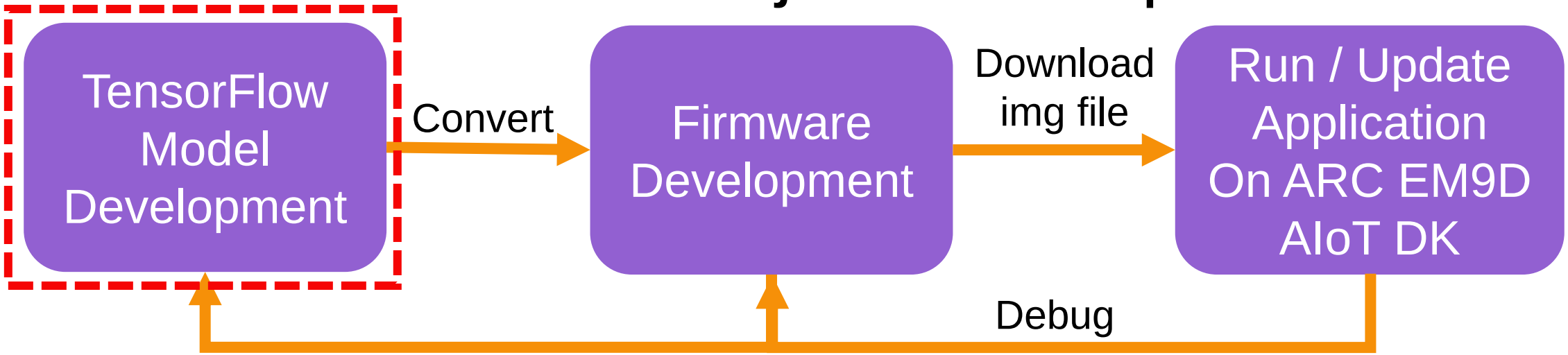
```
[9]:-128,
result:8
number
[0]:-128,
[1]:-128,
[2]:-128,
[3]:-128,
[4]:-128,
[5]:-128,
[6]:-128,
[7]:-128,
[8]:127,
[9]:-128,
result:8
```

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

What is TensorFlow?

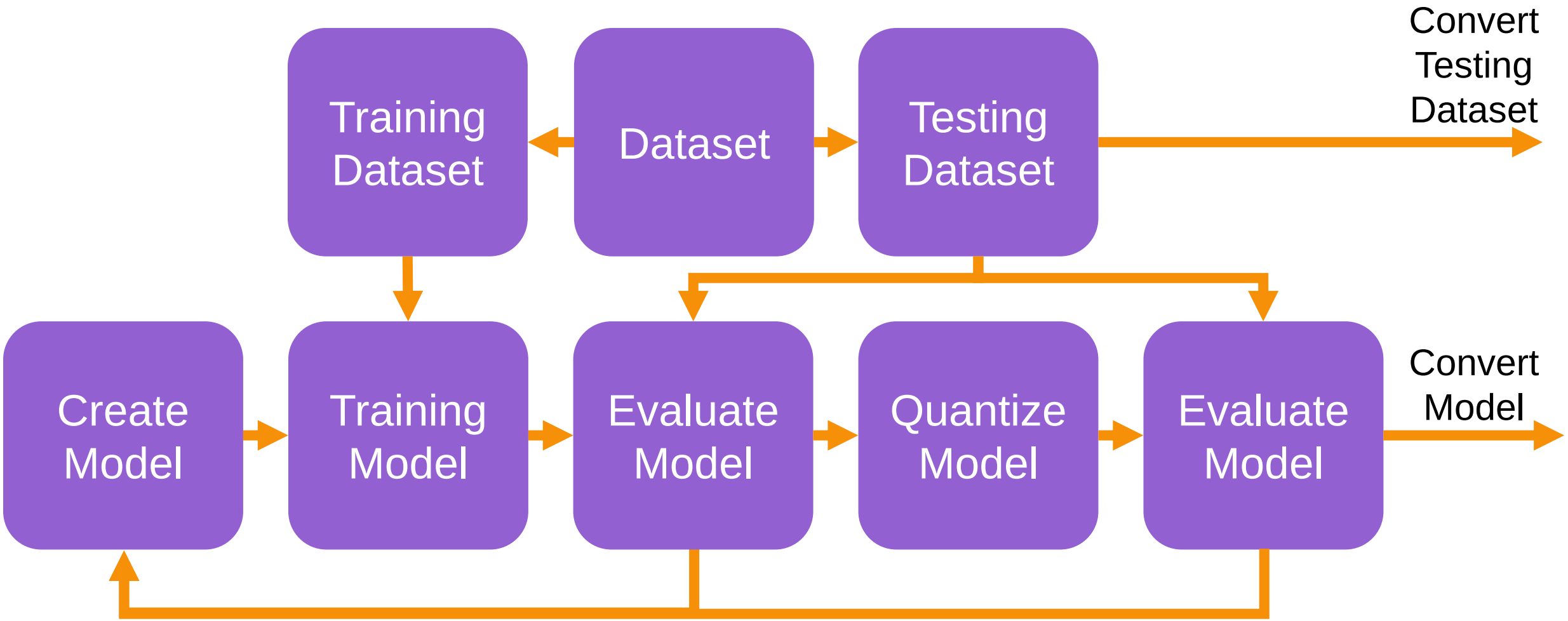
- Created by the Google Brain team, TensorFlow is an open source library for numerical computation and large-scale machine learning.
- TensorFlow bundles together a slew of machine learning, deep learning (aka neural networking) models, algorithms and makes them useful by way of a common metaphor.
-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

ARC EM9D AIoT DK Project Development Flow

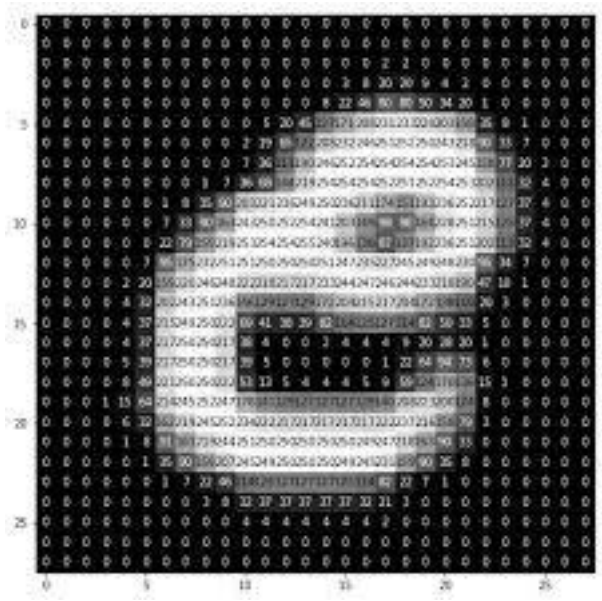


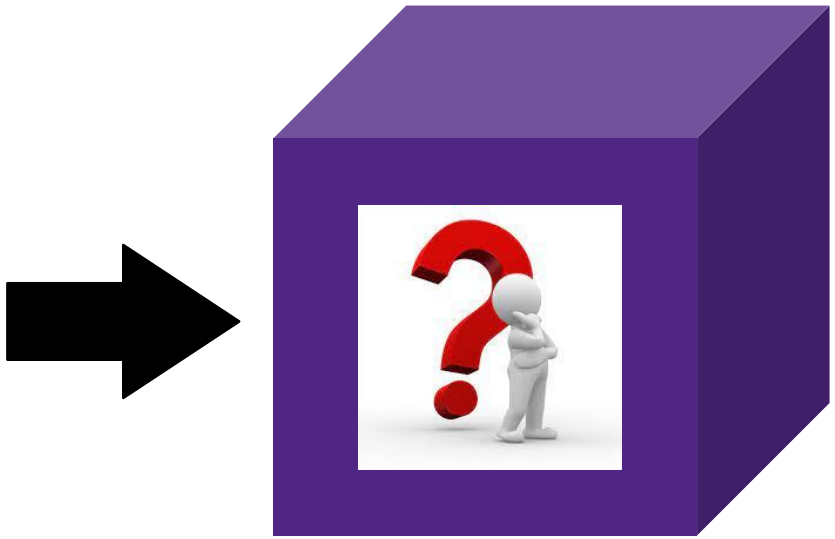
Stage	TensorFlow Model Development	Firmware Development	Run / Update Application On ARC EM9D AIoT DK
Tool	Anaconda Cygwin	Cygwin Metaware or ARC GNU VirtualBox (Ubuntu 20.04)	JTAG Himax-FT4222-GUI USB Cable
Language	Python 3	C language C++ language	

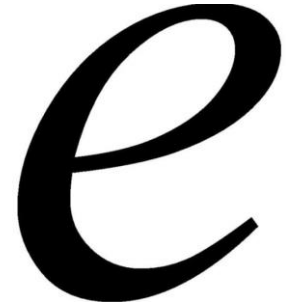
TensorFlow Model Development



Lab5: EMNIST Letters Recognition (Training)







Input

Model

Answer

Lab5: EMNIST Letters Recognition (Training)

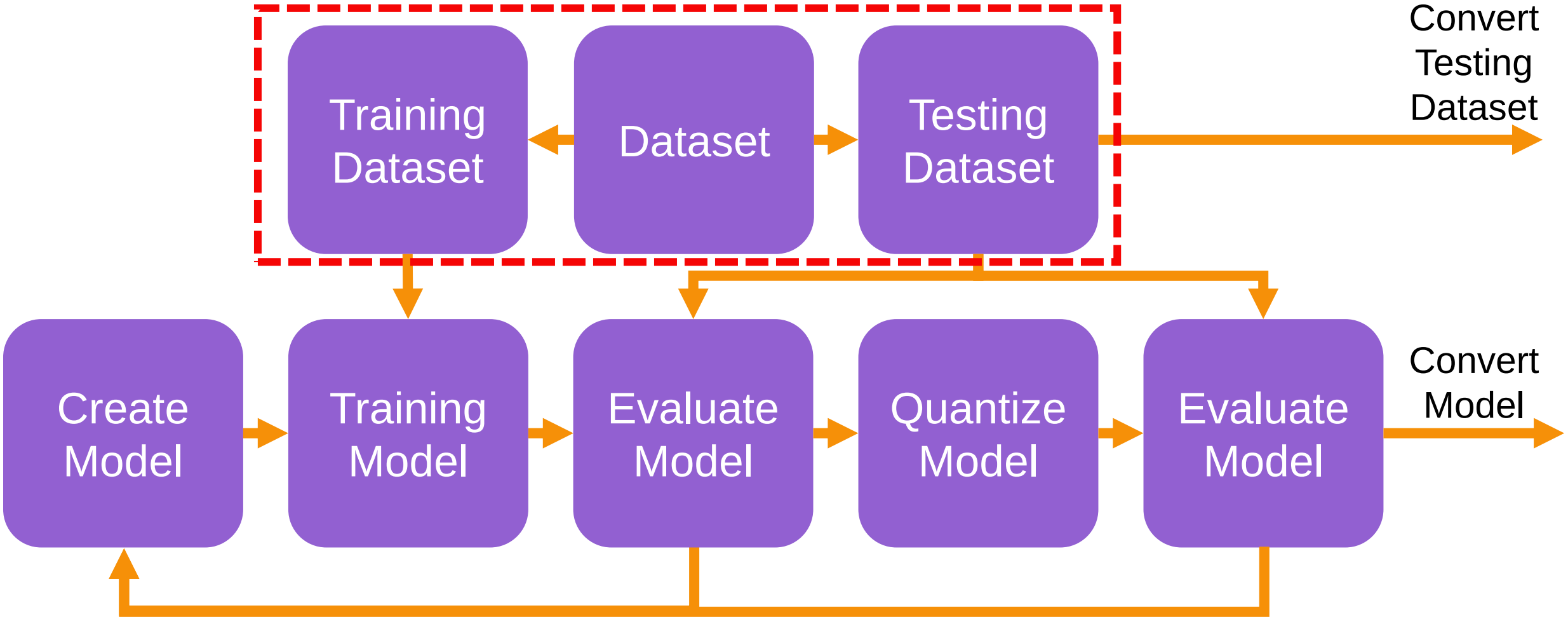
- Open Jupyter Notebook (TensorFlow)
- Go to “Lab5_tflm_emnist_training_letter/”
- Open “Lab5_tflm_emnist_training.ipynb”

Lab5: EMNIST Letters Recognition (Training)

- Import module and function you need to build your model

```
import matplotlib.pyplot as plt
import numpy as np

import tensorflow as tf
import tensorflow_datasets as tfds
import tensorflow.keras as keras
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 and save to the dataset_buffer

train_images_database, train_labels_database = tfds.as_numpy(tfds.load(
    'emnist/letters',
    split = 'train',
    shuffle_files = False,
    batch_size = -1,
    as_supervised = True,
))

test_images_database, test_labels_database = tfds.as_numpy(tfds.load(
    'emnist/letters',
    split = 'test',
    shuffle_files = False,
    batch_size = -1,
    as_supervised = True,
))
```

letters

digits

Emnist Dataset

Lab5: EMNIST Letters Recognition (Training)

- Error during download dataset (When PC doesn't have this dataset)

  If your PC can't download this dataset, please refer to Appendix-5.

```
# Import training and testing dataset and save to the dataset_buffer

train_images_database, train_labels_database = tfds.as_numpy(tfds.load(
    'emnist/letters',
    split = 'train',
    shuffle_files = False,
    batch_size = -1,
    as_supervised = True,
))
```

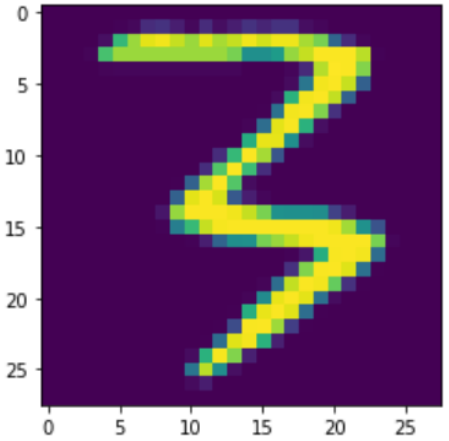
```
    --> 519             raise ConnectionError(e, request=request)
        520
        521         except ClosedPoolError as e:

    ConnectionError: HTTPSConnectionPool(host='www.itl.nist.gov', port=443): Max retries exceeded with url: /iaui/vip/cs_link
    s/EMNIST/gzip.zip (Caused by NewConnectionError('<urllib3.connection.HTTPSConnection object at 0x000001EBD426FE50>: Faile
    d to establish a new connection: [Errno 11001] getaddrinfo failed'))
```

Lab5: EMNIST Letters Recognition (Training)

- Dataset transport

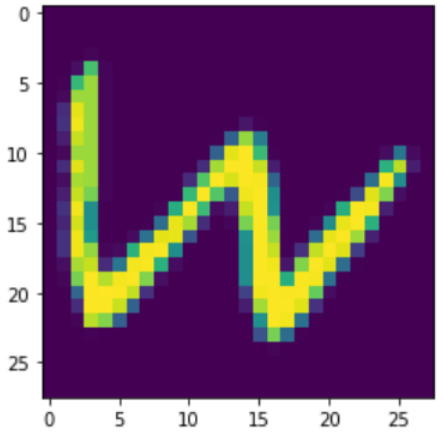
```
# Transport row/col of image
for img_index in range(0, train_images_database.shape[0]):
    for channel_index in range(0, train_images_database.shape[3]):
        train_images_database[img_index, :, :, channel_index] =  train_images_database[img_index, :, :, channel_index].transp

# Transport row/col of image
for img_index in range(0, test_images_database.shape[0]):
    for channel_index in range(0, test_images_database.shape[3]):
        test_images_database[img_index, :, :, channel_index] =  test_images_database[img_index, :, :, channel_index].transpos
```

Before



Transport

After



Lab5: EMNIST Letters Recognition (Training)

- Dataset preprocessing

```
num_classes = np.amax(train_labels_database);
total_train_image = train_images_database.shape[0];
total_test_image = test_images_database.shape[0];

# Make class numbering start at 0
train_labels_database = train_labels_database - 1
test_labels_database = test_labels_database - 1

img_rows = 28
img_cols = 28
img_channel = 1
input_shape = (img_rows, img_cols, img_channel)

train_images_database = train_images_database.reshape([train_images_database.shape[0], img_rows, img_cols, img_channel])
test_images_database = test_images_database.reshape([test_images_database.shape[0], img_rows, img_cols, img_channel])
```

Train: Reshape from 88800*28*28 to 88800*28*28*1

Test: Reshape from 14800*28*28 to 14800*28*28*1

Lab5: EMNIST Letters Recognition (Training)

- Dataset preprocessing

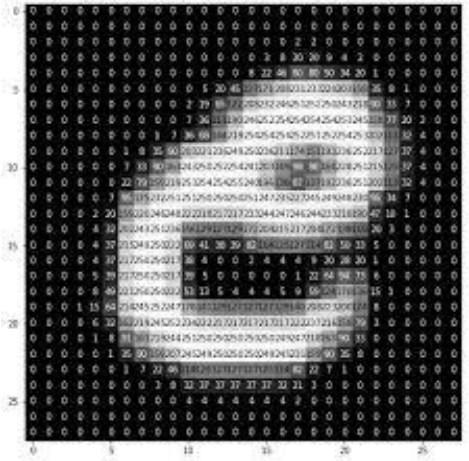
```
# Import training and testing from dataset_buffer
train_images = train_images_database
train_labels = train_labels_database

test_images = test_images_database
test_labels = test_labels_database
```

```
# Dataset preprocessing #1

# Transfer to nparray
train_images = train_images.astype('float32')
train_labels = to_categorical(train_labels, num_classes, dtype = 'float32')
test_images = test_images.astype('float32')
test_labels = to_categorical(test_labels, num_classes, dtype = 'float32')
```

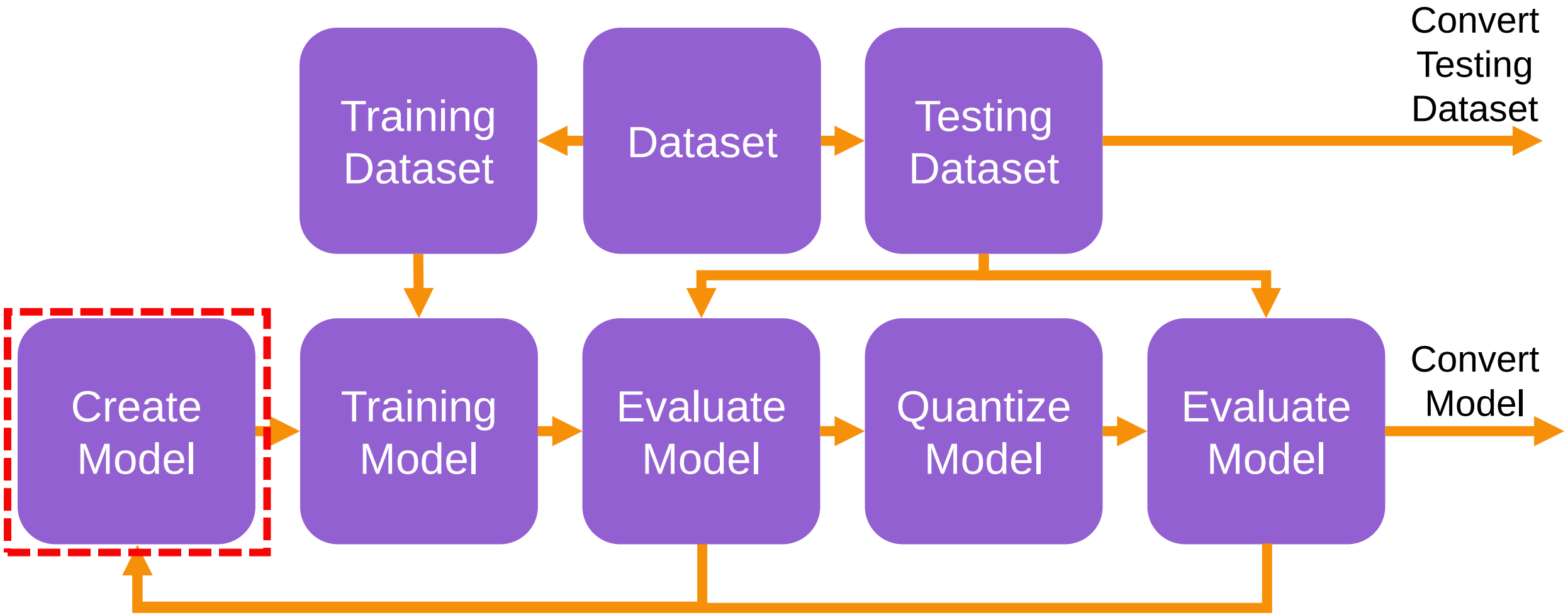
Lab5: EMNIST Letters Recognition (Training)

- 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
```



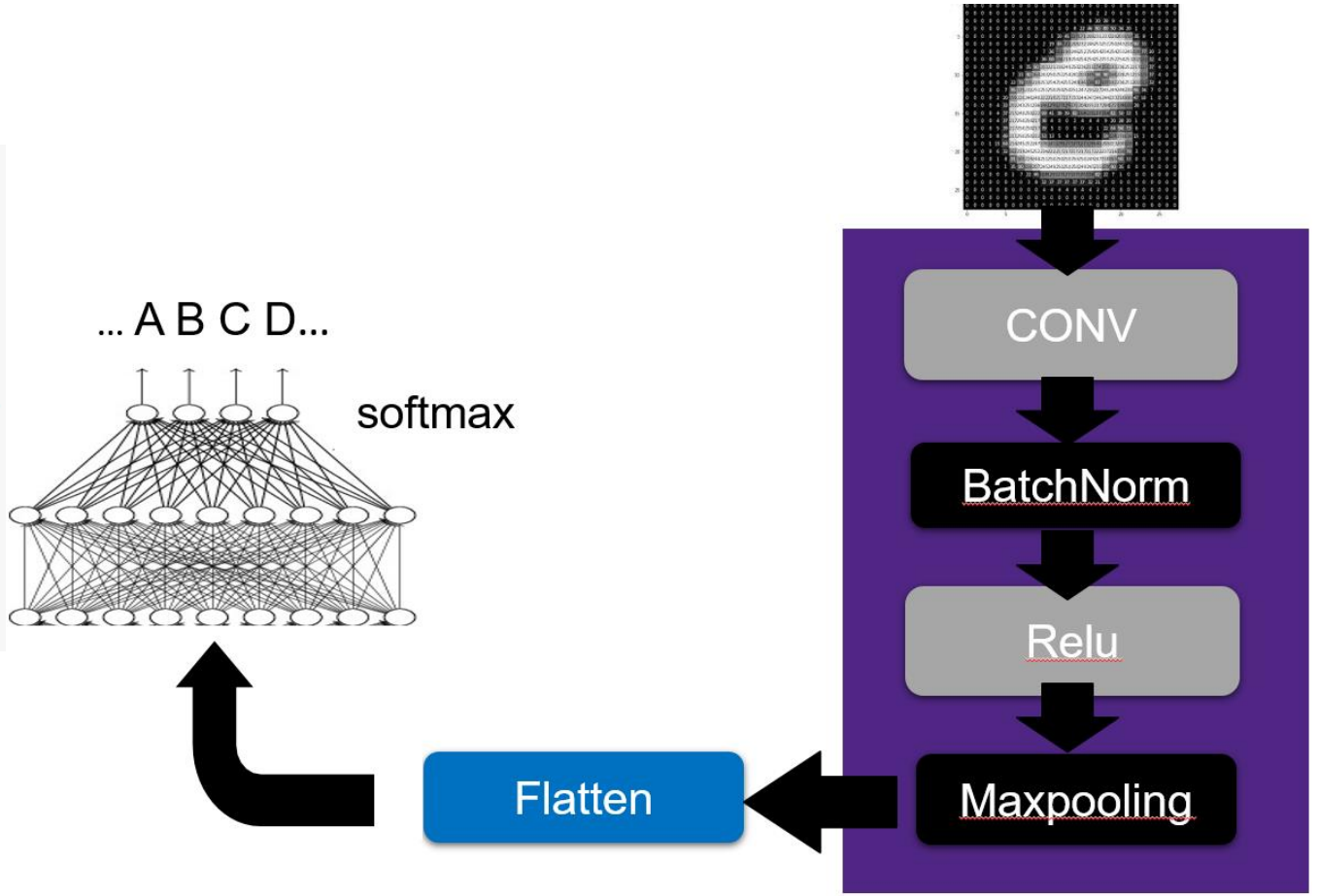
Lab5: EMNIST Letters Recognition (Training)



Lab5: EMNIST Letters Recognition (Training)

- Model Create

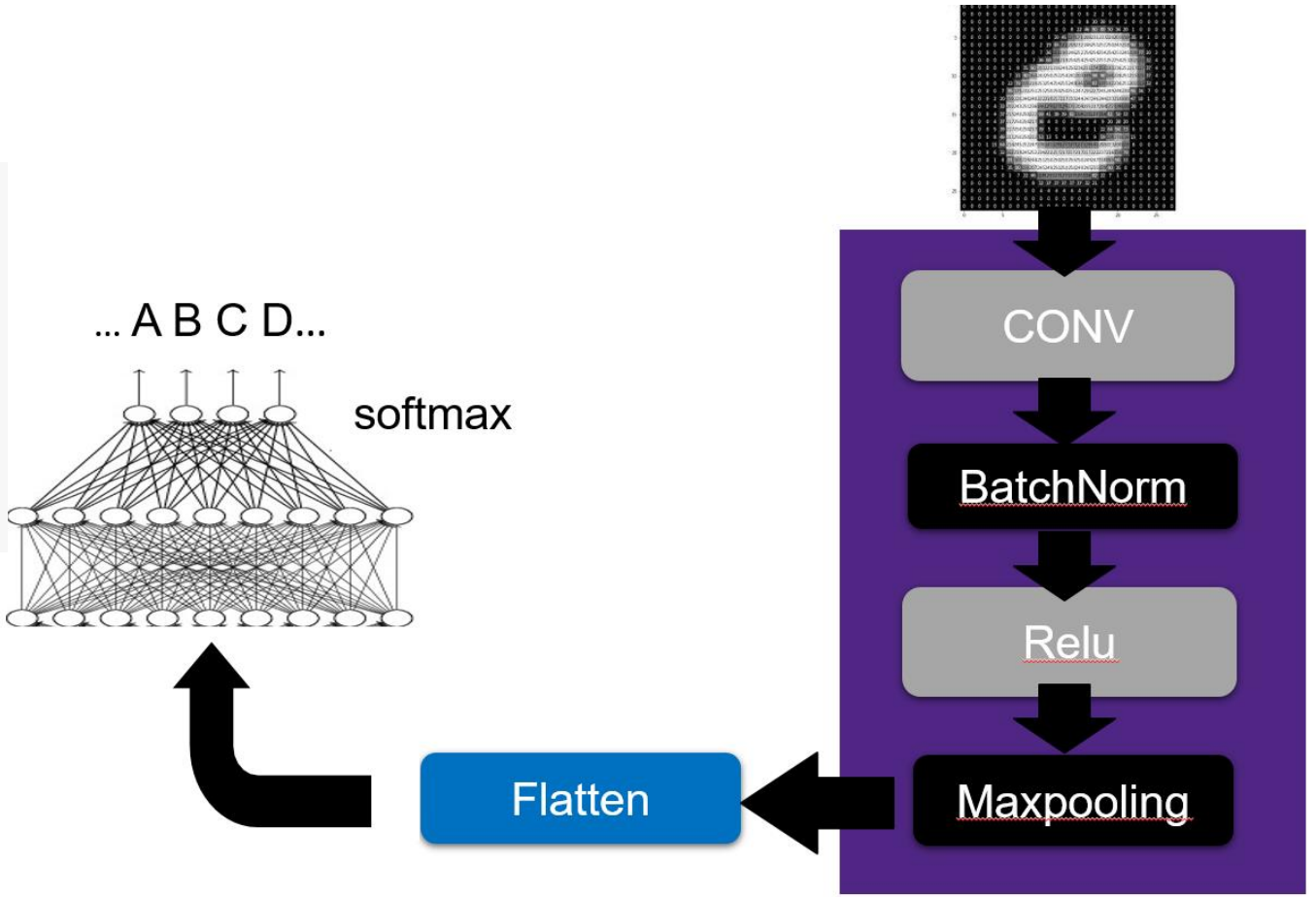
```
# Model create #1

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

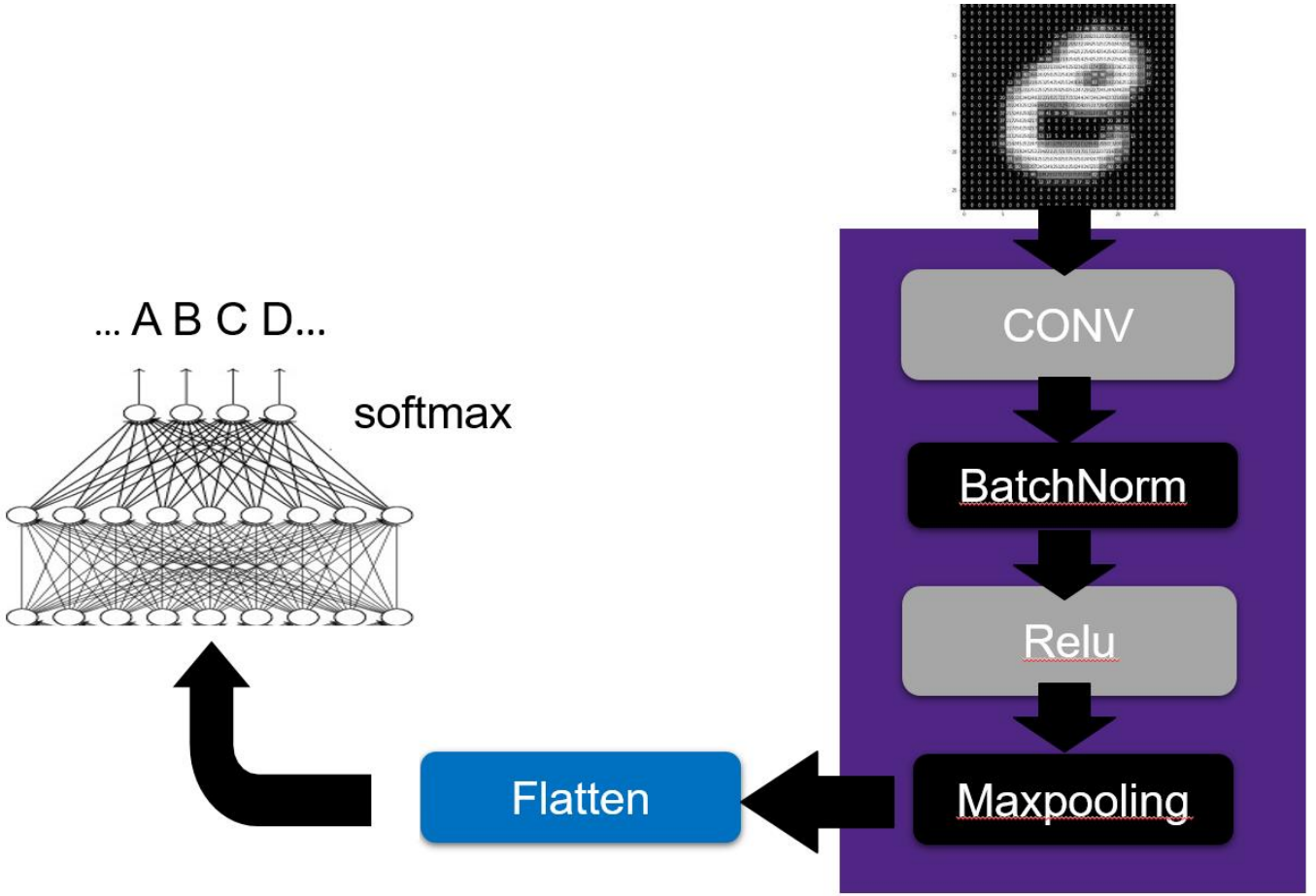
```
#Conv3
model.add(Conv2D(filters=32,
                 kernel_size=(filter_x, filter_y),
                 padding="same",
                 input_shape=input_shape))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(MaxPooling2D())
```



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)

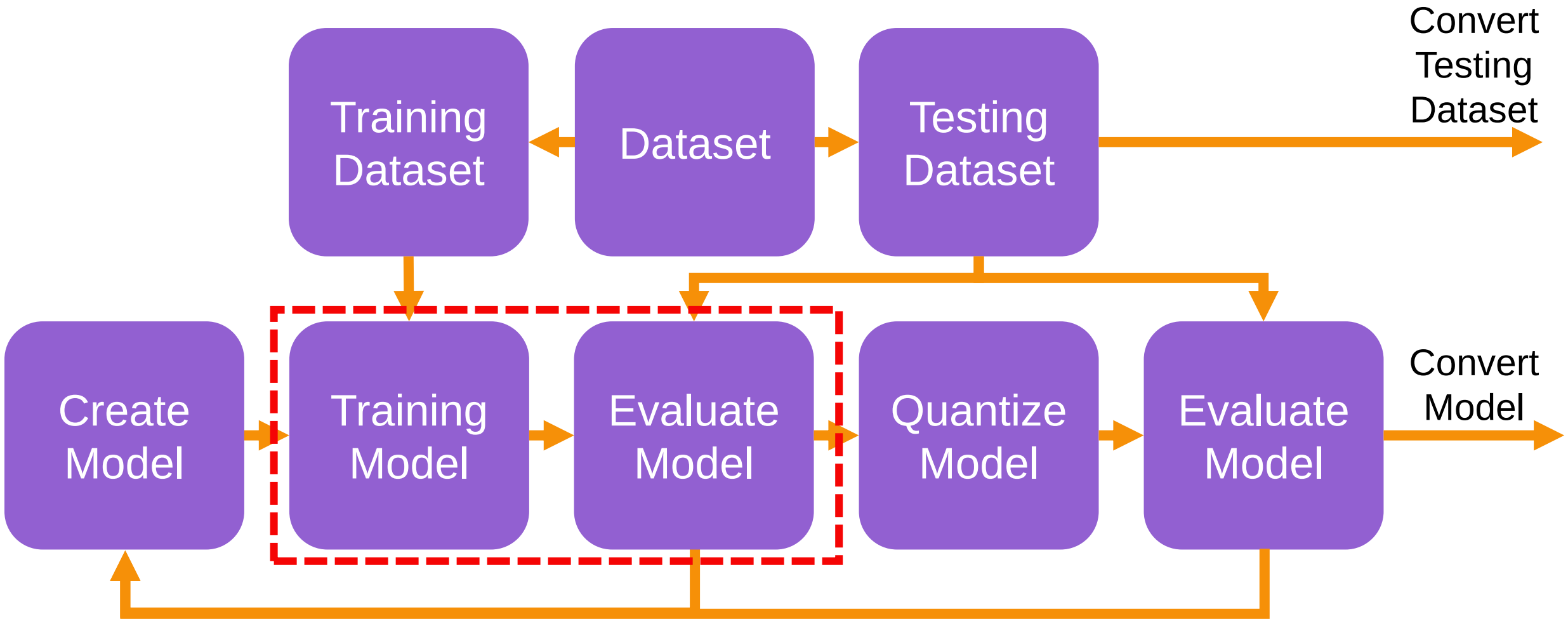
- Show model

```
# 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 0.30948373675346375
accuracy 0.896486461625671
```

Lab5: EMNIST Letters Recognition (Training)

- Save and load model weights (Only weights, without model)

```
# 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)

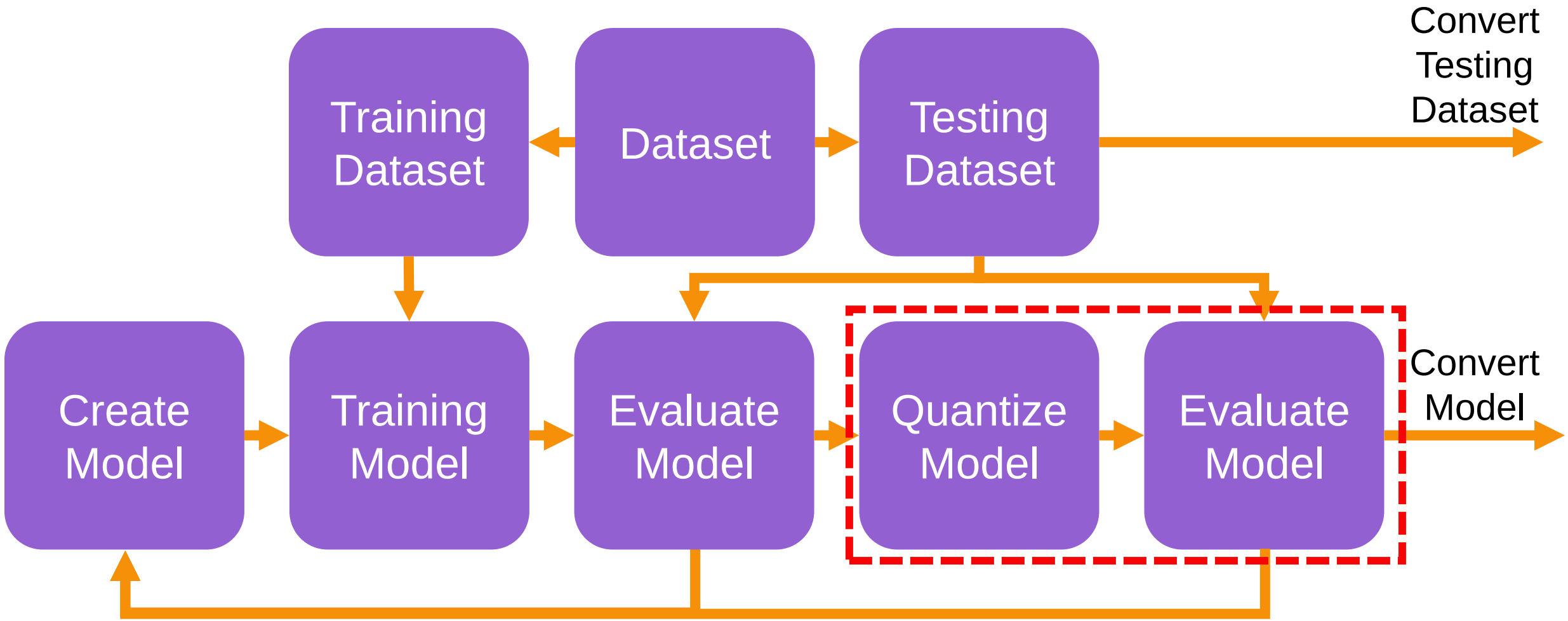
- Save and load all model(Both weights and model)

```
# Save model and weights of this model
model.save('model_save')
```

```
# LOAD model and weights of this model
model_2 = keras.models.load_model('model_save')

# Model Evaluation
score = model_2.evaluate(test_images, test_labels, verbose = 0)
print('test loss', score[0])
print('accuracy', score[1])
```

```
test loss 0.30948373675346375
accuracy 0.896486461625671
```

Lab5: EMNIST Letters Recognition (Training)



Lab5: EMNIST Letters Recognition (Training)

- Reload and preprocess images

```
# Import training and testing from dataset_buffer
test_images = test_images_database
test_labels = test_labels_database

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

test_images = thinning(test_images)
test_images = (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)

- Decide the testing sample size

  you can reduce 1.00 → 0.01 to speed up the evaluation

Evaluate TensorFlow Lite INT-8 Model

Full test set contains 14800 samples. Evaluating int8 model on it might take more than 10 minutes. If you want to get estimation faster, please, limit number of samples to be evaluated by reducing **max_samples** value

In [30]:
```
max_samples = int(test_images_database.shape[0] * 0.20)
# max_samples = int(test_images_database.shape[0] * 1.00)

print(max_samples)
```

```
2960
```

Lab5: EMNIST Letters Recognition (Training)

- Loading TensorFlow Lite Model

```
import pathlib

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

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

```
# 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']
```

Lab5: EMNIST Letters Recognition (Training)

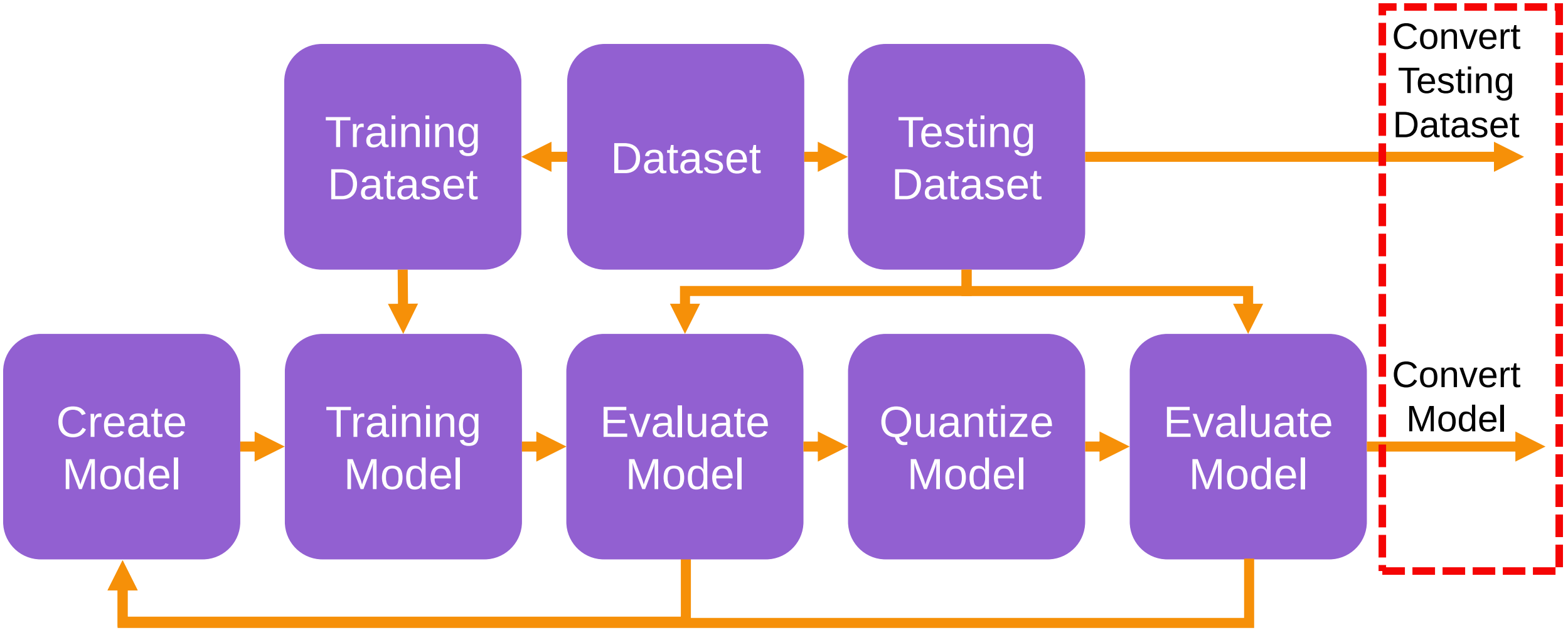
- Evaluate TensorFlow Lite Model

```
# Import training and testing from dataset_buffer
tflm_test_images = test_images_database
ans_test_labels = test_labels_database

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

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

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

```
91.21621621621621%
```

Lab5: EMNIST Letters Recognition (Training)



Lab5: EMNIST Letters Recognition (Training)

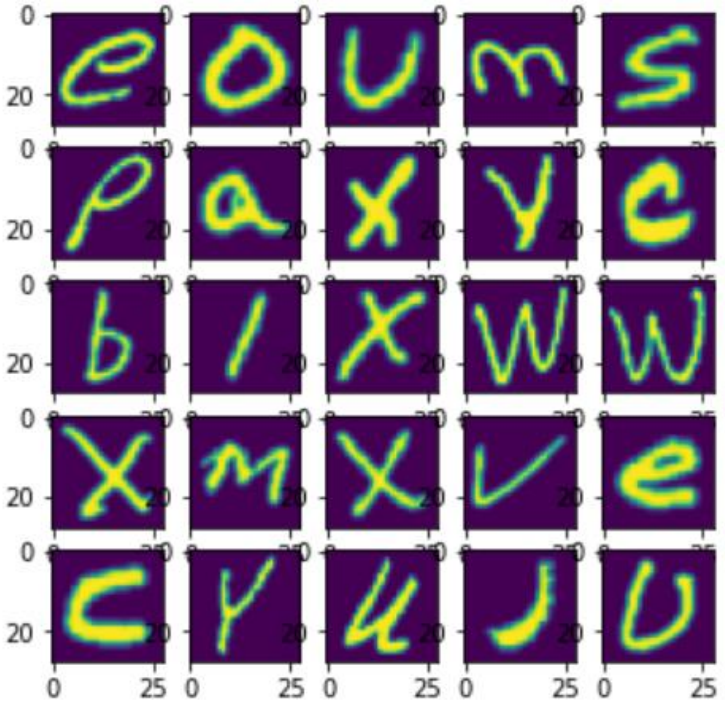
- Convert testing dataset to cpp file

```
import random

# Import training and testing from dataset_buffer
test_images = test_images_database
test_labels = test_labels_database

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

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 cpp file

```
fig=plt.figure(figsize=(10, 10))
rows = 5
cols = 5

for index in range(0, num_of_samples):
    img = test_images[random_test_images[index]]
    fig.add_subplot(rows, cols, (index + 1))
    plt.imshow(img)

plt.show()
```



Lab5: EMNIST Letters Recognition (Training)

- Convert testing dataset to cpp file

```
samples_file = open("generated/test_samples.cpp", "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.cpp”

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

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

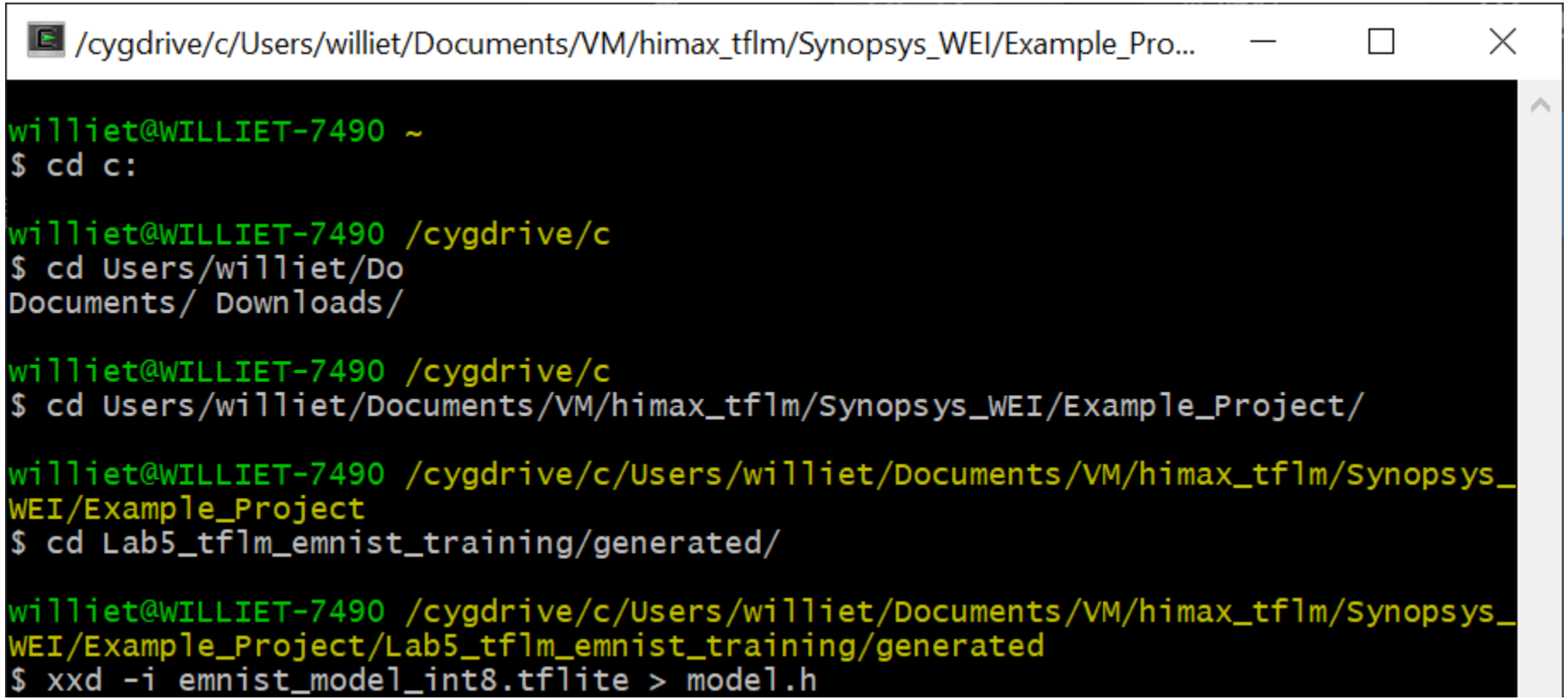
Lab5: EMNIST Letters Recognition (Training)

- Open cygwin64 and convert tflite to c file

```
$ cd c:
$ cd Users/{username}/{Workshop_path}
$ cd Lab5_tflm_emnist_training_number/generated/
$ xxd -i emnist_model_int8.tflite > model.h
```

Lab5: EMNIST Letters Recognition (Training)

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



Lab5: EMNIST Letters Recognition (Training)

Conclusion

- Import all the libraries as tools to build a model
- Load dataset and split it into training set and testing set
- Preprocess your dataset to a format that your model accepts
- 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 ARC

Lab5: Integrate TensorFlow Lite and ARC

- Integrated Project: Lab5_emnist_letter_test

- Copy “Lab5_tflm_emnist_training_letter/generated/model.h” to “Lab5_emnist_letter_test/inc/model.h“

- Copy “Lab5_tflm_emnist_ training_letter/generated/test_samples.cpp” to “Lab5_emnist_letter/src/test_samples.cpp “

Lab5: Integrate TensorFlow Lite and ARC

- 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 ARC

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

```
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 ARC

- Select for your model (tflitemicro_algo.cpp)

  Please confirm the model array in model.h is same as you use.

model.h

```
unsigned char emnist_model_int8_tflite[] = {
  0x20, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x3
  0x00, 0x00, 0x12, 0x00, 0x1c, 0x00, 0x04, 0x0
  0x10, 0x00, 0x14, 0x00, 0x00, 0x00, 0x18, 0x0
  0x03, 0x00, 0x00, 0x00, 0x5c, 0x08, 0x01, 0x0
  0xf4, 0xea, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x0
  0x01, 0x00, 0x00, 0x00, 0x0c, 0x00, 0x00, 0x0
  0x04, 0x00, 0x08, 0x00, 0x08, 0x00, 0x00, 0x0
  0x17, 0x00, 0x00, 0x00, 0x13, 0x00, 0x00, 0x0
```

tflitemicro_algo.cpp

```
extern "C" int tflitemicro_algo_init()
{
    int ercode = 0;
    TfLiteStatus ret;

    error_reporter = &micro_error_reporter;

    error_reporter->Report("TFLM model setting\n");
    model = ::tflite::GetModel(emnist_model_int8_tflite);
    if (model->version() != TFLITE_SCHEMA_VERSION) {
        error_reporter->Report(
            "Model provided is schema version %d not equal "
            "to supported version %d.\n",
            model->version(), TFLITE_SCHEMA_VERSION);
    }
```

Lab5: Integrate TensorFlow Lite and ARC

- Edit micro_op_resolver for your model (tflitemicro_algo.cpp)

  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();
```

```
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 ARC

- Edit micro_op_resolver for your model (tflitemicro_algo.cpp)

```
// 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();
```

```
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 ARC

- You can find usable 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 ARC

- Declare TensorFlowlite input and output buffer in “tflitemicro_algo.cpp”

```
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 ARC

- Make sure your Tensor area size is large enough in “tflitemicro_algo.cpp”

```
// 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 ARC

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

```
for(test_j = 0; test_j < kImageSize; test_j ++)
    input_buf[test_j] = (test_samples[test_i].image[test_j] <= 210) ? -128 : 127;
```

- Start invoking (running model)

```
test_result = tflitemicro_algo_run(&input_buf[0]);
```

Lab5: Integrate TensorFlow Lite and ARC

- Return max element from output[0] to output[25]
- Print out Test_Answer and TFLM_Inference_Answer
- Calculate probability

```
sprintf(uart_buf, "Ans_Num: %2d, TFLM_Num: %2d\r\n", test_samples[test_i].label, test_result);
uart0_ptr->uart_write(uart_buf, strlen(uart_buf));
    board_delay_ms(10);

sprintf(uart_buf, "Ans_Char: %s, TFLM_Char: %s\r\n\n", kCategoryLabels[test_samples[test_i].label], kCategoryLabels[test_result]);
uart0_ptr->uart_write(uart_buf, strlen(uart_buf));
    board_delay_ms(10);

if(test_samples[test_i].label == test_result)
    test_correct ++;
```

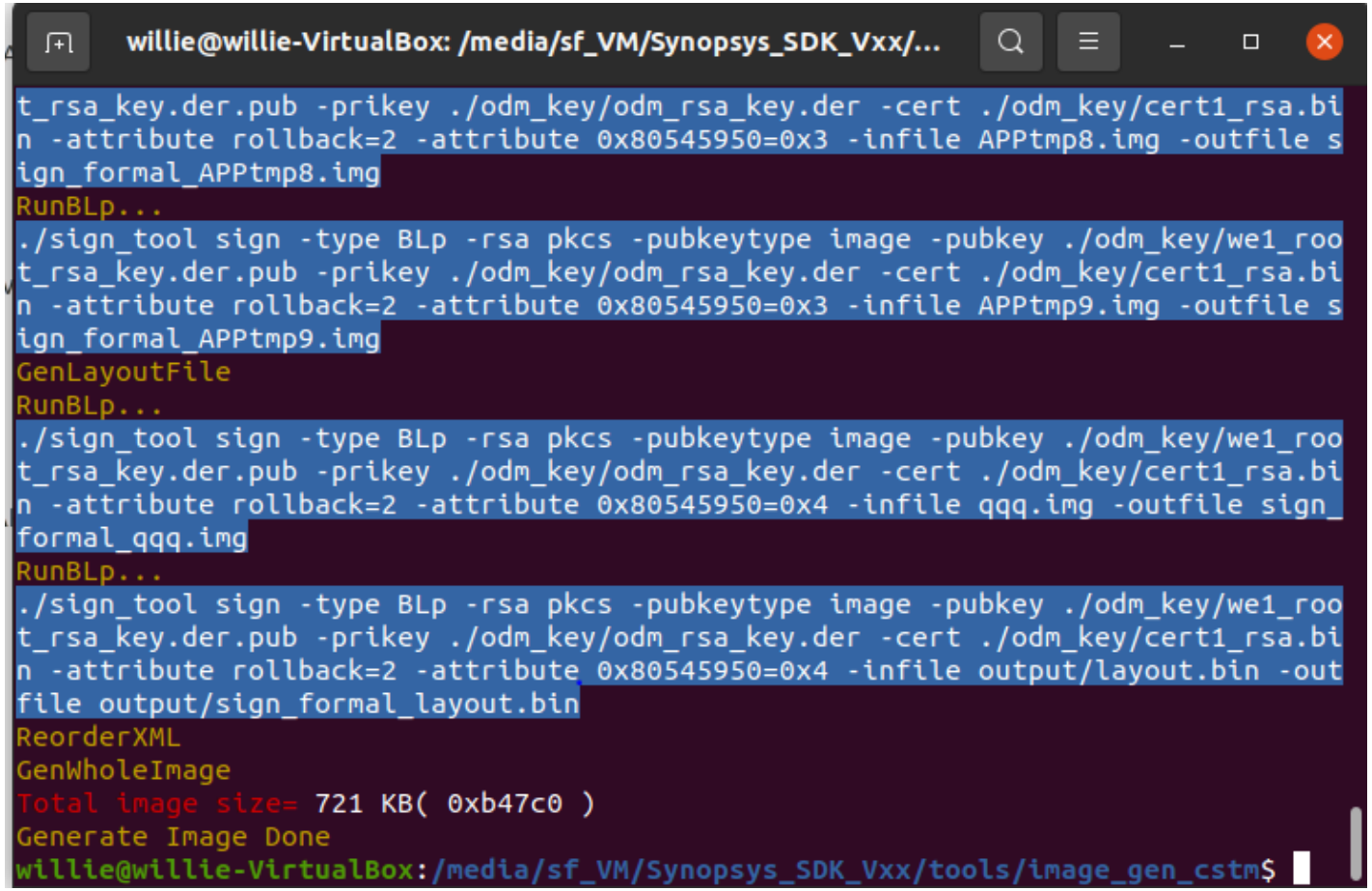
Lab5: Integrate TensorFlow Lite and ARC

Conclusion

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

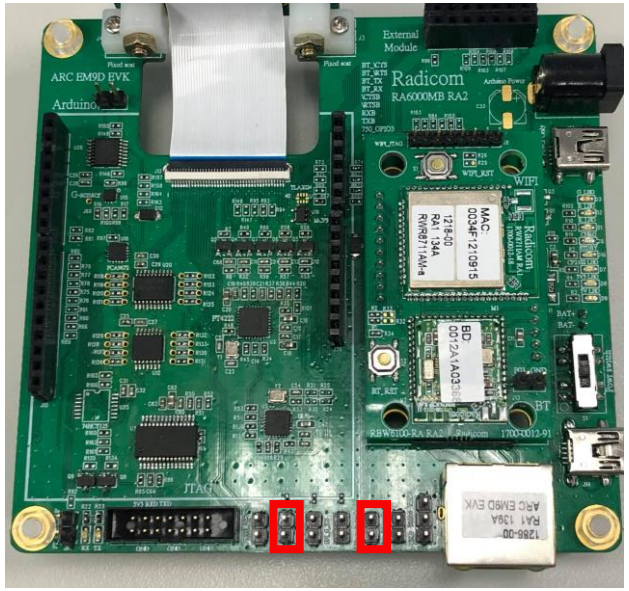
Lab5: Integrate TensorFlow Lite and ARC

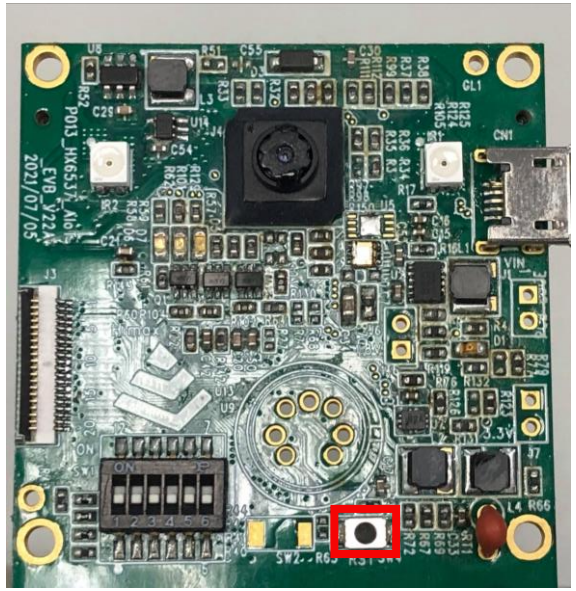
- Make project “Lab5_tflm_emnist_letter” and convert to the image file



Lab5: Integrate TensorFlow Lite and ARC

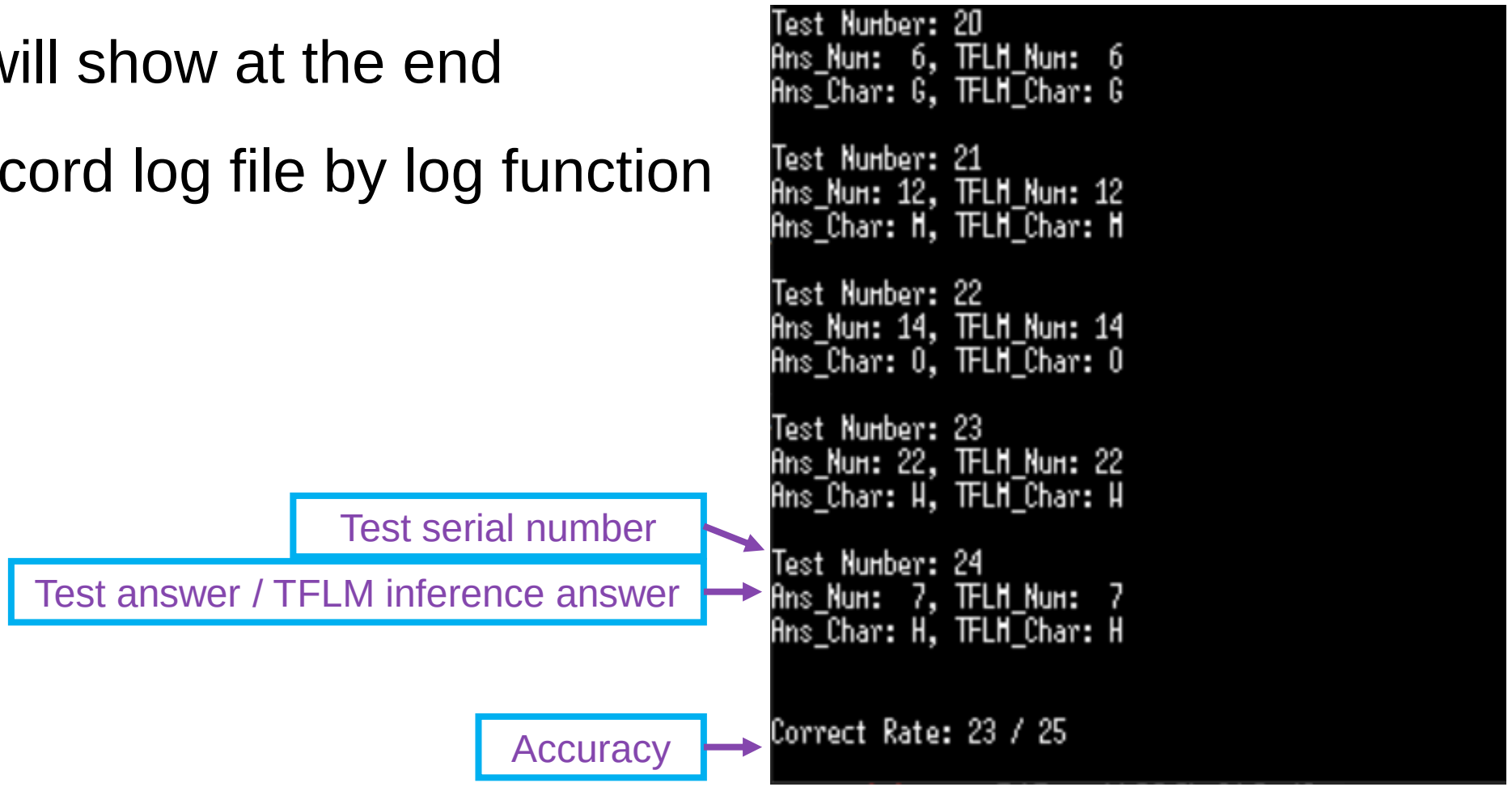
1. Short J20 and J11 for update application mode
2. Download image file to CPU
3. Open J20 for run mode
4. Press reset button SW4. MCU will reset and run the application





Lab5: Integrate TensorFlow Lite and ARC

- 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



Appendix-5: Troubleshooting – EMNIST Dataset Install

Troubleshooting – EMNIST Dataset Install

If you can’t download EMNIST dataset and know someone has already downloaded, you can copy their dataset any copy to your PC manually.

1. Go to “Jupyter notebook root path” in the PC which has this dataset.
   Default: “C:\Users\{username}\”
2. Copy folder ”tensorflow_datasets” to another PC’s “Jupyter notebook root path”.