Convolutional neural networks

Practice Session 9

Changho Suh

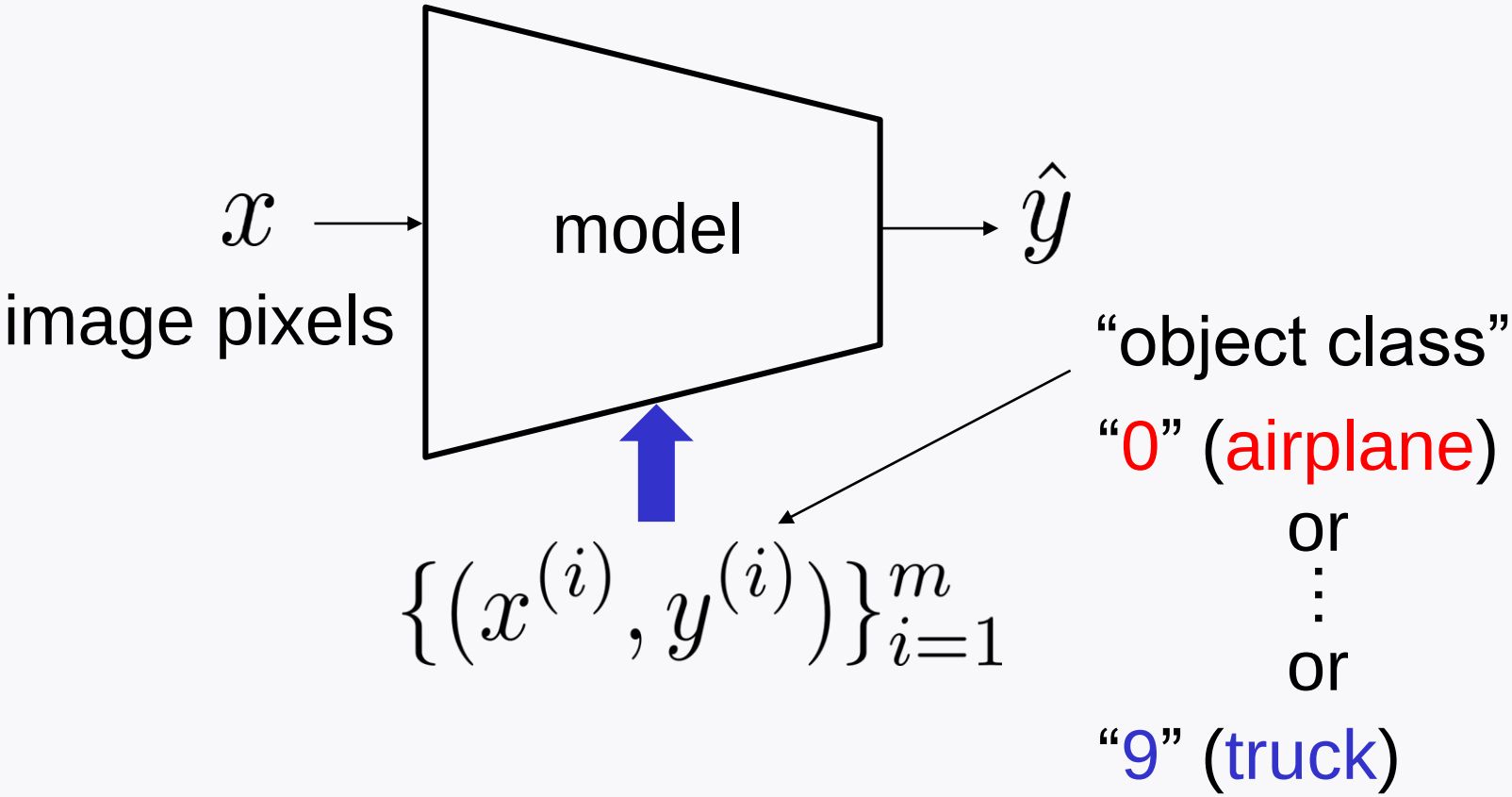
January 24, 2024

Outline

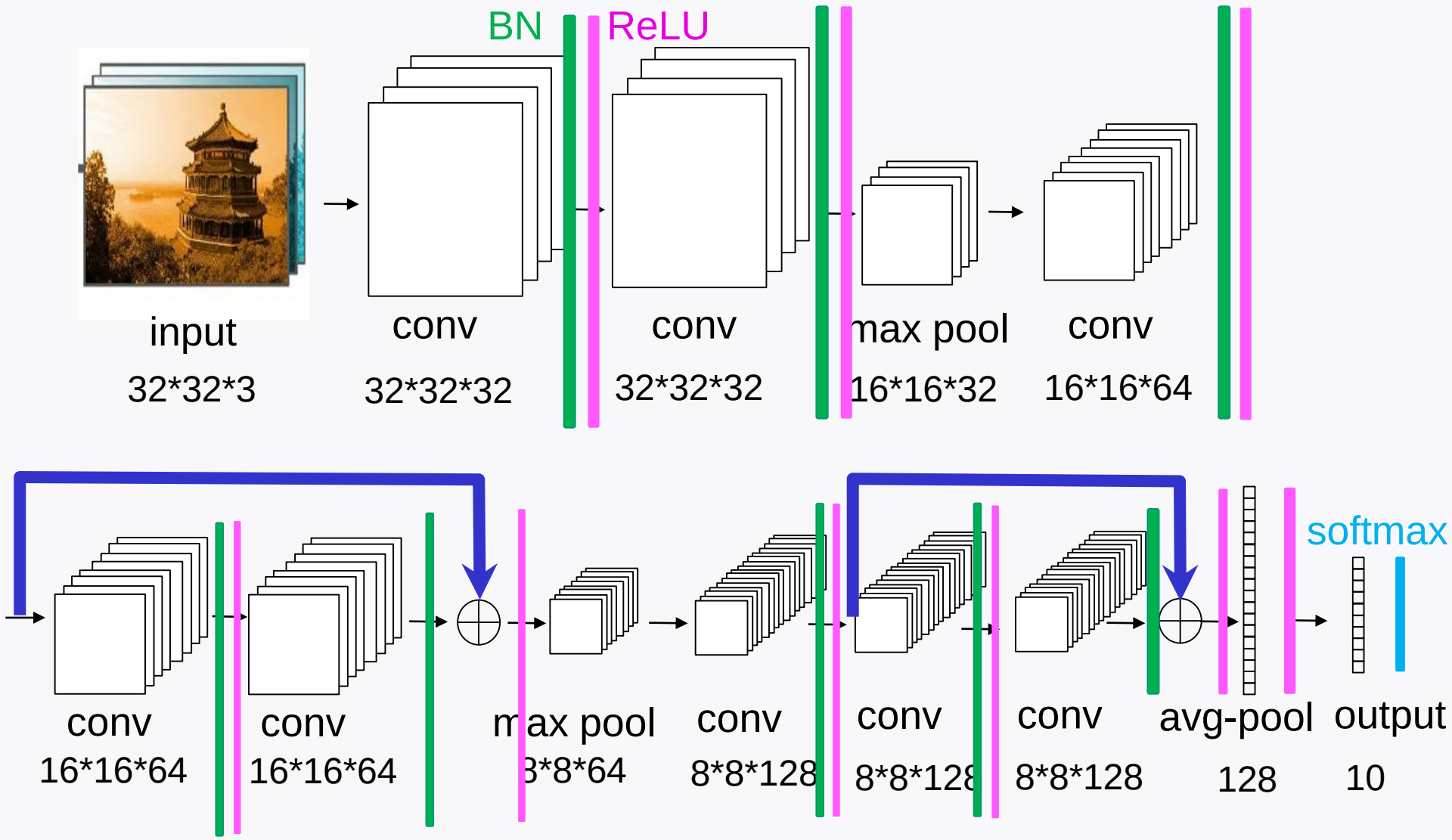
Will implement the simplified ResNet.

Task: Image recognition (CIFAR10)

Image recognition



Simplified ResNet



Two ways to construct a model

1. **Sequential**:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense

Model_NN = Sequential()

Model_NN.add(Flatten(input_shape=(28,28)))

Model_NN.add(Dense(128, activation='relu'))

Model_NN.add(Dense(10, activation='softmax'))
```

Two ways to construct a model

2. **Model**:

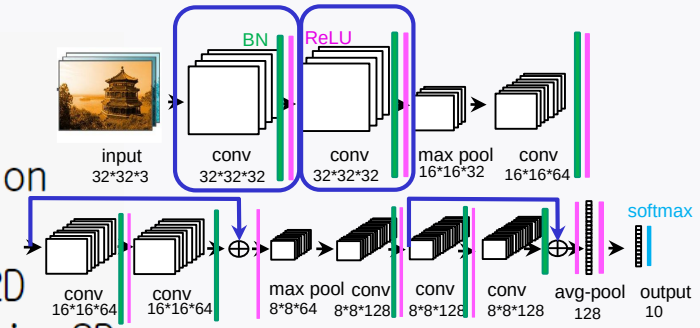
```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input

inputs = Input(shape=(28,28))
flatten = Flatten()(inputs)
hidden = Dense(128, activation='relu')(flatten)
outputs = Dense(10, activation='softmax')(hidden)

model_NN = Model(inputs = inputs, outputs = outputs)
```

ResNet: Tensorflow coding

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import ReLU
from tensorflow.keras.layers import Conv2D, MaxPool2D
from tensorflow.keras.layers import Add, AveragePooling2D
```




```
inputs = Input(shape=(32,32,3))
x = Conv2D(32, kernel_size=(3,3), strides=(1,1),
           padding='same', use_bias=False)(inputs)
x = BatchNormalization()(x)
x = ReLU()(x)
```

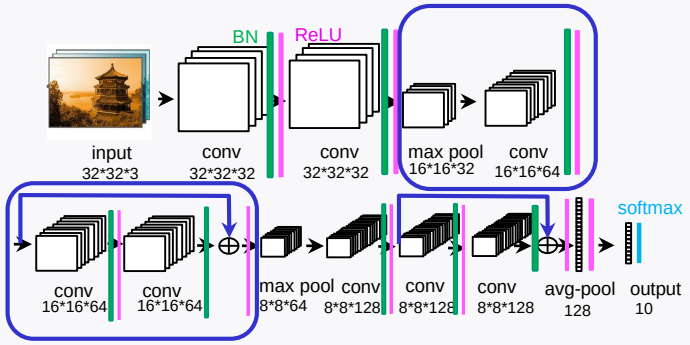
```
x = Conv2D(32, kernel_size=(3,3), strides=(1,1),
           padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = ReLU()(x)
```

ResNet: Tensorflow coding

```
x = MaxPool2D(pool_size=(2,2), strides=(2,2),
              padding='valid')(x)
x = Conv2D(64, kernel_size=(3,3), strides=(1,1),
           padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = ReLU()(x)
```

```
# 1st skip connection
skip = x
x = Conv2D(64, kernel_size=(3,3), strides=(1,1),
           padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = ReLU()(x)
x = Conv2D(64, kernel_size=(3,3), strides=(1,1),
           padding='same', use_bias=False)(x)
x = BatchNormalization()(x)

x = Add()([x, skip])
x = ReLU()(x)
```

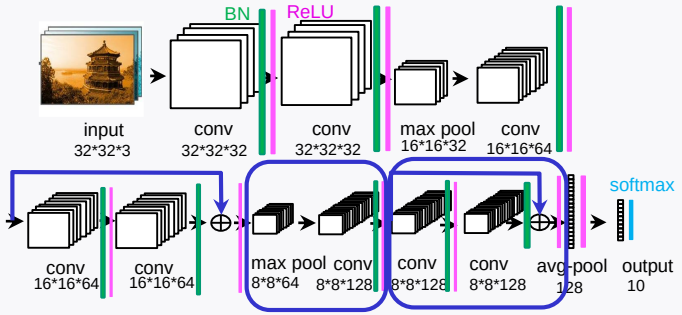


ResNet: Tensorflow coding

```
x = MaxPool2D(pool_size=(2,2), strides=(2,2),
              padding='valid')(x)
x = Conv2D(128, kernel_size=(3,3), strides=(1,1),
           padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = ReLU()(x)
```

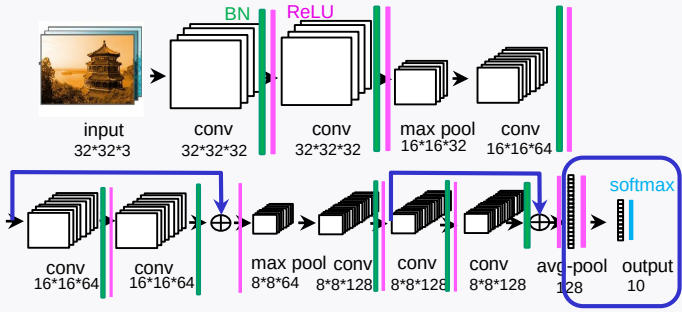
```
# 2nd skip connection
skip = x
x = Conv2D(128, kernel_size=(3,3), strides=(1,1),
           padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = ReLU()(x)
x = Conv2D(128, kernel_size=(3,3), strides=(1,1),
           padding='same', use_bias=False)(x)
x = BatchNormalization()(x)

x = Add()([x, skip])
x = ReLU()(x)
```



ResNet: Tensorflow coding

```
# Average Pooling
x = AveragePooling2D(pool_size=(8,8))(x)
x = Flatten()(x)
x = ReLU()(x)
outputs = Dense(10, activation='softmax')(x)
```




```
Model_resnet = Model(inputs = inputs, outputs = outputs)
```

Compile

```
from tensorflow.keras.optimizers import Adam

opt = Adam(learning_rate = 0.001,
           beta_1 = 0.9,
           beta_2 = 0.999)

Model_resnet.compile(optimizer = opt,
                     loss='sparse_categorical_crossentropy',
                     metrics=['acc'])
```

Training & evaluation

```
# training
Model_resnet.fit(X_train,y_train,epochs=20)
```

```
# Evaluation
test_performance = Model_resnet.evaluate(X_test,y_test)
print(test_performance)
```

```
[0.7550032138824463, 0.8246999979019165]
```