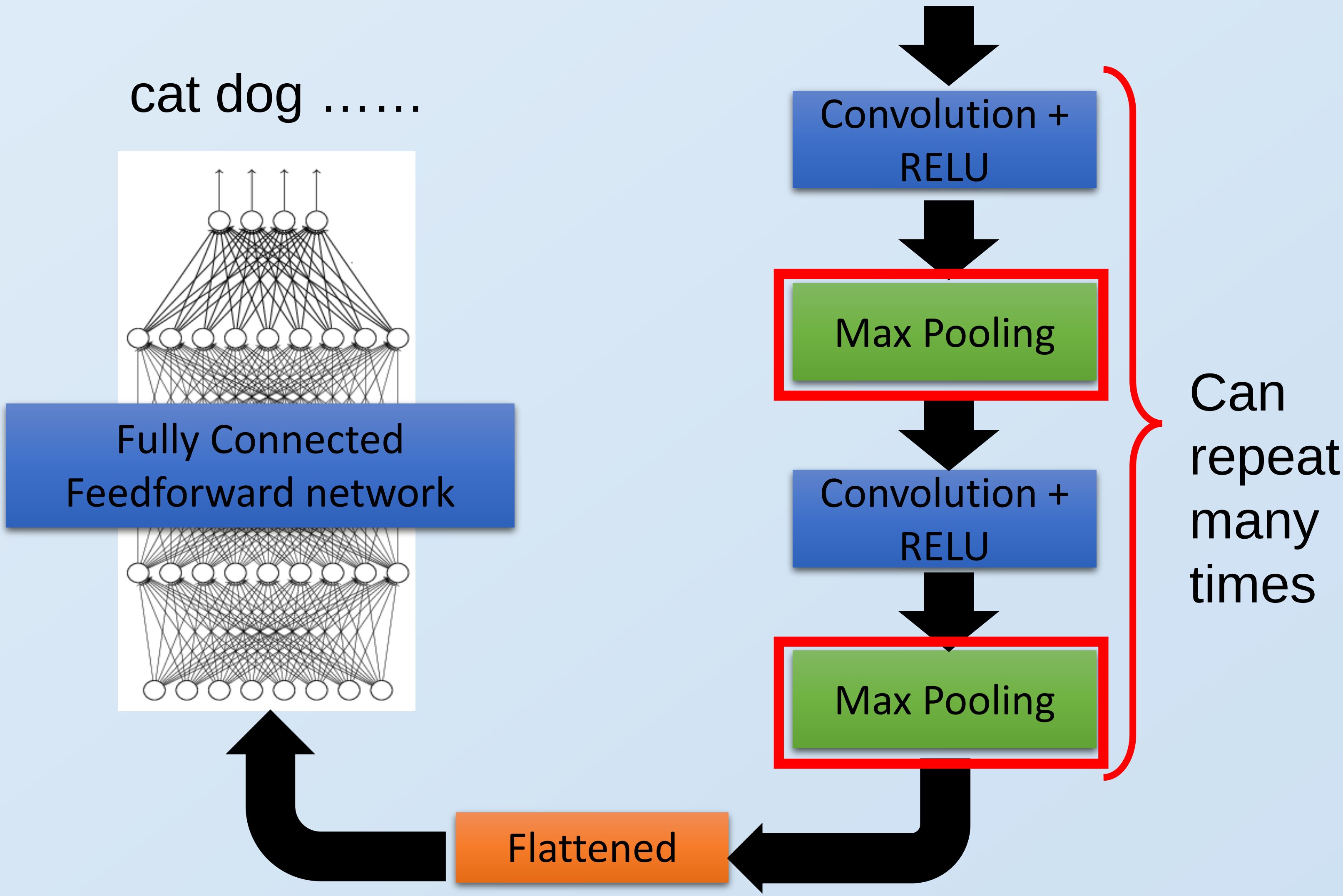
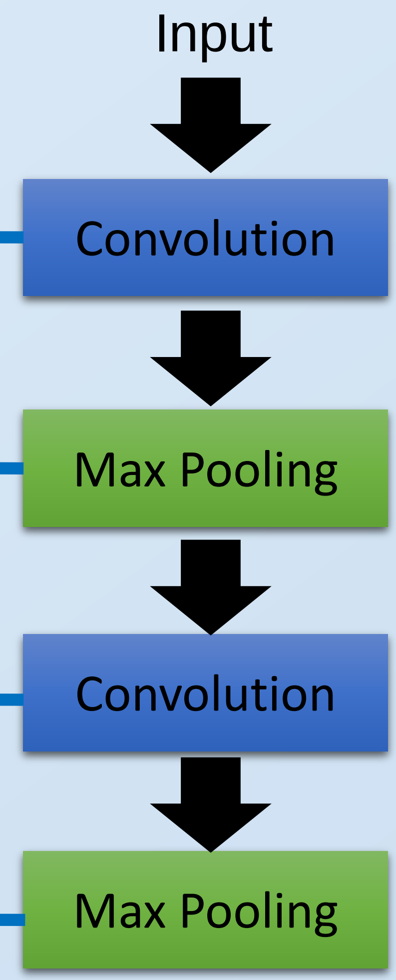
Convolutional Neural Networks

The whole CNN



CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D array)***



1 x 28 x 28

```
model2.add( Convolution2D( 25,3,3,
        input_shape=(28,28,1)) )
```

How many parameters for each filter? 9

25 x 26 x 26

```
model2.add(MaxPooling2D((2,2)))
```

25 x 13 x 13

```
model2.add(Convolution2D(50,3,3))
```

How many parameters for each filter? 225= 25x9

50 x 11 x 11

```
model2.add(MaxPooling2D((2,2)))
```

50 x 5 x 5

Convolutional Neural Networks using keras and tensorflow

```
import tensorflow as tf
from tensorflow.keras.layers import *
from keras.models import Model
from keras.optimizers import Adam, SGD

# Define CNN model
def create_model():
  model = tf.keras.models.Sequential()
  model.add(tf.keras.layers.Conv2D(32, (3, 3),
      input_shape=(28,28,1), activation='relu'))
  model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
  model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
  model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2),
      padding='same'))
  model.add(tf.keras.layers.Flatten())
  model.add(tf.keras.layers.Dense(64,activation ='relu'))
  model.add(tf.keras.layers.Dense(10,activation = 'softmax'))
  return model
```

Convolutional Neural Networks using keras and tensorflow

```
# Read Data
(x_train, y_train), (x_test, y_test) =
     tf.keras.datasets.mnist.load_data()
x_train = np.expand_dims(x_train, -1)/255
x_test = np.expand_dims(x_test, -1)/255

# Convert y to one-hot
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

model = create_model()
model.summary() # Prints model structure
```

Convolutional Neural Networks using keras and tensorflow

```
model.summary() # Prints model structure
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_4 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 13, 13, 32)        0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 11, 11, 64)        18496     
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 6, 6, 64)          0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 2304)              0         
_________________________________________________________________
dense_4 (Dense)              (None, 64)                147520    
_________________________________________________________________
dense_5 (Dense)              (None, 10)                650       
=================================================================
Total params: 166,986
Trainable params: 166,986
Non-trainable params: 0
```

Convolutional Neural Networks using keras and tensorflow

```
model = create_model()
model.summary() # Prints model structure

# Compile model – Build computation graph
model.compile(loss="categorical_crossentropy",
      optimizer="adam", metrics=["accuracy"])

# Fit model and store results in history object
history = model.fit(x_train, y_train, batch_size=128, epochs=20,
      validation_data=(x_test, y_test))
```

Convolutional Neural Networks using keras and tensorflow

```
# Read Data
(x_train, y_train), (x_test, y_test) =
     tf.keras.datasets.mnist.load_data()
x_train = np.expand_dims(x_train, -1)/255
x_test = np.expand_dims(x_test, -1)/255

# Convert y to one-hot
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

model = create_model()
model.summary() # Prints model structure

model.compile(loss="categorical_crossentropy",
      optimizer="adam", metrics=["accuracy"])

# Fit model and store results in history object
history = model.fit(x_train, y_train, batch_size=128, epochs=20,
      validation_data=(x_test, y_test))
```

Convolutional Neural Networks using keras and tensorflow

```
# Fit model and store results in history object
history = model.fit(x_train, y_train, batch_size=128, epochs=10,
      validation_data=(x_test, y_test))
```

Epoch 1/10 469/469 [==============================] - 43s 93ms/step - loss: 0.2240 - accuracy: 0.9348 - val_loss: 0.0599 - val_accuracy: 0.9820

Epoch 2/10 469/469 [==============================] - 43s 92ms/step - loss: 0.0591 - accuracy: 0.9821 - val_loss: 0.0449 - val_accuracy: 0.9860

Epoch 3/10 469/469 [==============================] - 43s 92ms/step - loss: 0.0430 - accuracy: 0.9868 - val_loss: 0.0379 - val_accuracy: 0.9868

Epoch 4/10 469/469 [==============================] - 43s 92ms/step - loss: 0.0331 - accuracy: 0.9898 - val_loss: 0.0374 - val_accuracy: 0.9881

Epoch 5/10 469/469 [==============================] - 43s 92ms/step - loss: 0.0260 - accuracy: 0.9919 - val_loss: 0.0346 - val_accuracy: 0.9889

Epoch 6/10 469/469 [==============================] - 43s 92ms/step - loss: 0.0210 - accuracy: 0.9935 - val_loss: 0.0333 - val_accuracy: 0.9887

Epoch 7/10 469/469 [==============================] - 43s 92ms/step - loss: 0.0176 - accuracy: 0.9947 - val_loss: 0.0318 - val_accuracy: 0.9897

Epoch 8/10 469/469 [==============================] - 43s 92ms/step - loss: 0.0138 - accuracy: 0.9956 - val_loss: 0.0308 - val_accuracy: 0.9905

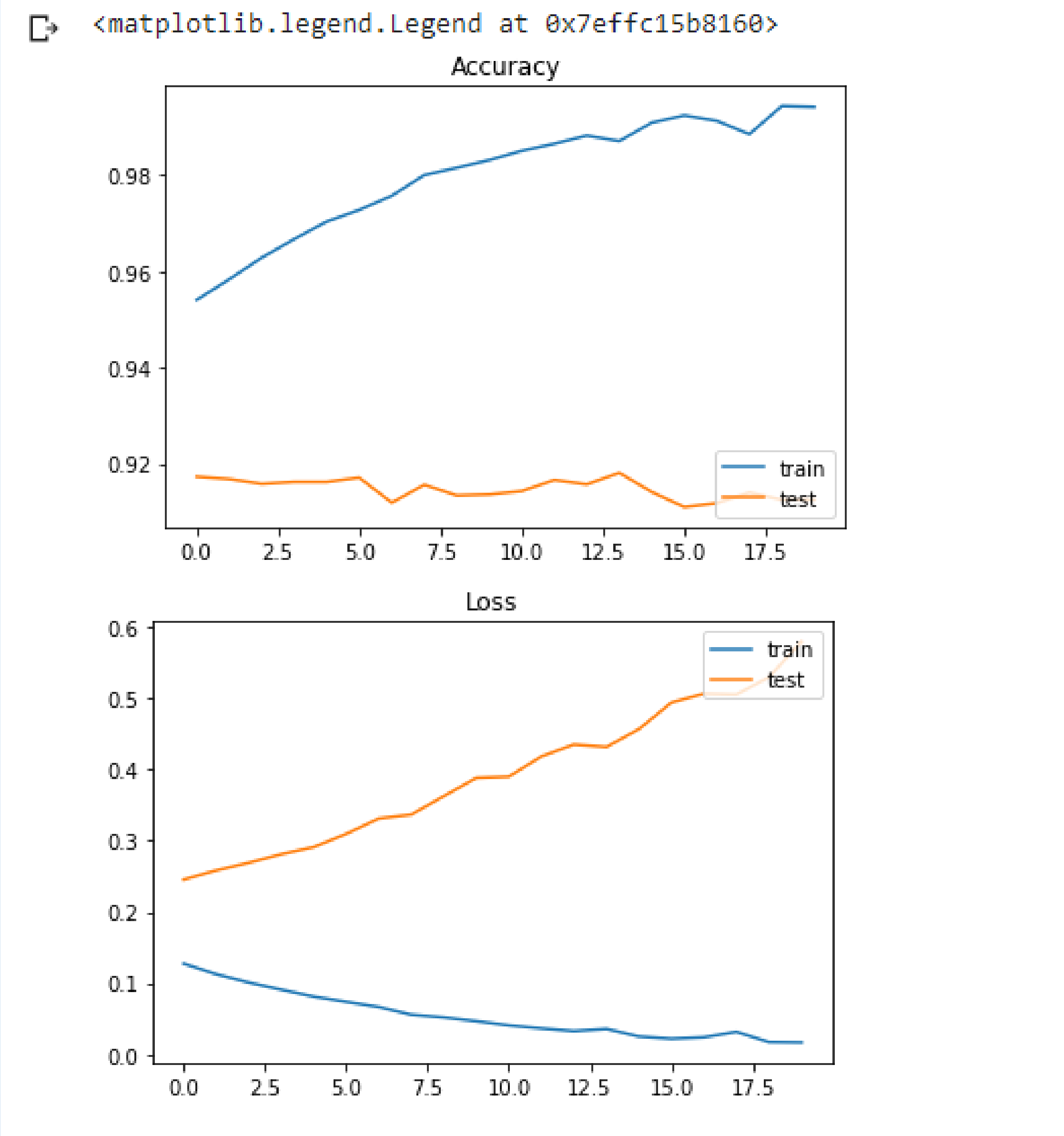
Epoch 9/10 469/469 [==============================] - 43s 92ms/step - loss: 0.0101 - accuracy: 0.9968 - val_loss: 0.0321 - val_accuracy: 0.9910

Epoch 10/10 469/469 [==============================] - 43s 92ms/step - loss: 0.0104 - accuracy: 0.9967 - val_loss: 0.0318 - val_accuracy: 0.9901

Convolutional Neural Networks using keras and tensorflow

```
# Plot loss and accuracy during training
fig, ax = plt.subplots()
ax.plot(history.history['accuracy'],label = 'train')
ax.plot(history.history['val_accuracy'],label = 'test')
ax.set_title('Accuracy')
ax.legend(loc='lower right')
fig, ax = plt.subplots()
ax.plot(history.history['loss'],label = 'train')
ax.plot(history.history['val_loss'],label = 'test')
ax.set_title('Loss')
ax.legend(loc='upper right')
```

Convolutional Neural Networks using keras and tensorflow



Improving your model

Changes in the architecture

- Add layers
- Add units in layers already present
- Modify strides – make sure you don’t waste computation by discarding information
- Problem – larger networks are prone to overfitting
- Convolution layers with non-unit stride as replacement for maxpooling

Activation functions

- Softmax usually works best for classification
- Relu, elu and Selu units work well for hidden layers

Improving your model

Regularization

- L1, L2 regularization – penalties for weight magnitudes; smaller weights are less likely to overfit
- Dropout – ignore some randomly-chosen units in every batch; make units learn independent features

Optimizer

- Adam tends to work best
- All optimizers are sensitive to initial learning rate

Batch size

- Smaller batches create more weight updates, but make training slower

Batch normalization

- Normalizes layer activations for every batch – tends to keep gradients in same value range during training