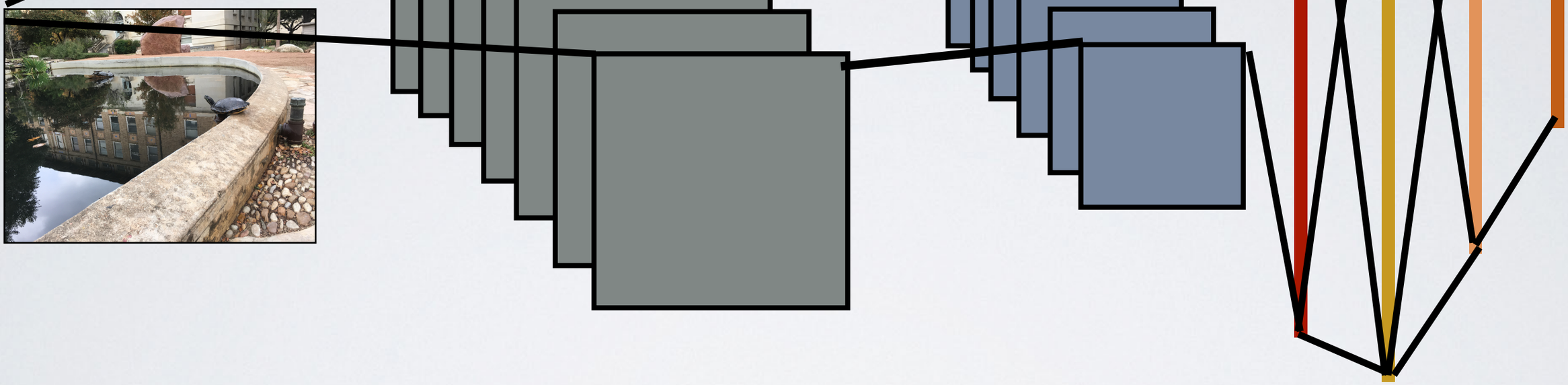

CONVOLUTIONAL NEURAL NETWORKS - II

Sumohana S. Channappayya

KEY REFERENCES

- https://www.tensorflow.org/tutorials/estimators/cnn

- https://www.tensorflow.org/tutorials/images/deep_cnn

- https://cs231n.github.io/convolutional-networks/

BUILDING A CNN USING TENSORFLOW

- Model architecture
- Creating and connecting the components
- Choosing the cost function
- Setting up data
- Training
- Validation

PROBLEM STATEMENT

Build a CNN-based handwritten digit classifier using the MNIST dataset (http://yann.lecun.com/exdb/mnist/)

MODEL ARCHITECTURE

- Input layer
- Two convolutional layers
- Two pooling layers (one per convolutional layer)
- One densely connected layer
- One logit layer

CREATING THE COMPONENTS

- **Input layer:** 28x28 image (MNIST dataset)
- **First convolutional layer:** 32 5x5 filters with ReLU activation
- **First pooling layer:** Max pooling 2x2, stride of 2
- **Second convolutional layer:** 64 5x5 filters with ReLU activation
- **Second pooling layer:** Max pooling 2x2, stride of 2
- **Dense layer:** 1024 nodes/neurons
- **Logit layer:** 10 nodes/neurons

CREATING AND CONNECTING THE COMPONENTS

- Input layer

```
input_layer = tf.reshape(features["x"], [-1, 28, 28, 1])
```

CREATING AND CONNECTING THE COMPONENTS

- Convolutional layer 1

```
conv1 = tf.layers.conv2d(inputs=input_layer, filters=32,
kernel_size=[5, 5], padding="same",
activation=tf.nn.relu)
```

CREATING AND CONNECTING THE COMPONENTS

- Pooling layer 1

```
pool1 = tf.layers.max_pooling2d(inputs=conv1,
pool_size=[2, 2], strides=2)
```

CREATING AND CONNECTING THE COMPONENTS

- Convolutional layer 2

```
conv2 = tf.layers.conv2d(inputs=pool1, filters=64,
kernel_size=[5, 5], padding="same",
activation=tf.nn.relu)
```

CREATING AND CONNECTING THE COMPONENTS

- Pooling layer 2

```
pool2 = tf.layers.max_pooling2d(inputs=conv2,
pool_size=[2, 2], strides=2)
```

CREATING AND CONNECTING THE COMPONENTS

- Dense layer

```
pool2_flat = tf.reshape(pool2, [-1, 7 * 7 * 64])

dense = tf.layers.dense(inputs=pool2_flat, units=1024,
activation=tf.nn.relu)
```

CREATING AND CONNECTING THE COMPONENTS

- Dropout layer

```
dropout = tf.layers.dropout(inputs=dense, rate=0.4,
training=mode == tf.estimator.ModeKeys.TRAIN)
```

CREATING AND CONNECTING THE COMPONENTS

- Logit layer

```
logits = tf.layers.dense(inputs=dropout, units=10)
```

CREATING AND CONNECTING THE COMPONENTS

- Prediction (for evaluation and prediction)

```
predictions = {

"classes": tf.argmax(input=logits, axis=1),

"probabilities": tf.nn.softmax(logits,
name="softmax_tensor")   }
```

CHOOSING THE COST FUNCTION

- Cross entropy loss function

```
loss =
tf.losses.sparse_softmax_cross_entropy(labels=labels,
logits=logits)
```

CHOOSING THE OPTIMISATION FUNCTION

- Gradient descent optimiser

```
optimizer =
tf.train.GradientDescentOptimizer(learning_rate=0.001)
train_op = optimizer.minimize(loss=loss,
global_step=tf.train.get_global_step())
```

REPORTING ACCURACY

- Report accuracy during evaluation

```
eval_metric_ops = {“accuracy":
tf.metrics.accuracy(labels=labels,
predictions=predictions["classes"])}
```

SETTING UP DATA

- Load MNIST data

```
mnist = tf.contrib.learn.datasets.load_dataset("mnist")

train_data = mnist.train.images

train_labels = np.asarray(mnist.train.labels,
dtype=np.int32)

eval_data = mnist.test.images

eval_labels = np.asarray(mnist.test.labels, dtype=np.int32)
```

PUTTING IT TOGETHER

- Create the Estimator

```
mnist_classifier =
tf.estimator.Estimator(model_fn=cnn_model_fn,
model_dir="/tmp/mnist_convnet_model")
```

TRAINING

- Train the model using the estimator

```
train_input_fn =
tf.estimator.inputs.numpy_input_fn(x={"x": train_data},
y=train_labels, batch_size=100, num_epochs=None,
shuffle=True)
mnist_classifier.train(input_fn=train_input_fn,
steps=20000)
```

EVALUATION

- Evaluate the trained model

```
eval_input_fn =
tf.estimator.inputs.numpy_input_fn(x={"x": eval_data},
y=eval_labels, num_epochs=1,shuffle=False)

eval_results =
mnist_classifier.evaluate(input_fn=eval_input_fn)
print(eval_results)
```

DEMO

SUMMARY

- TensorFlow provides an easy-to-use open source software library for implementing ML algorithms
- Powerful features under-the-hood for CPU/GPU optimisation
- Great equaliser