An Introduction to

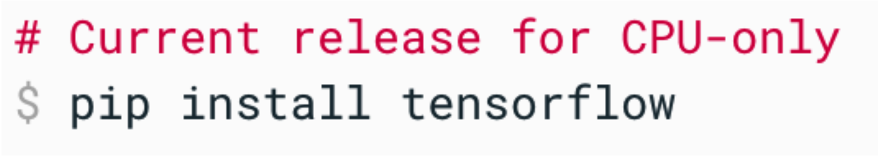
TensorFlow

Neural Networks & Deep Learning Course
Kamyar Ghajar
Fall 2018

Installation

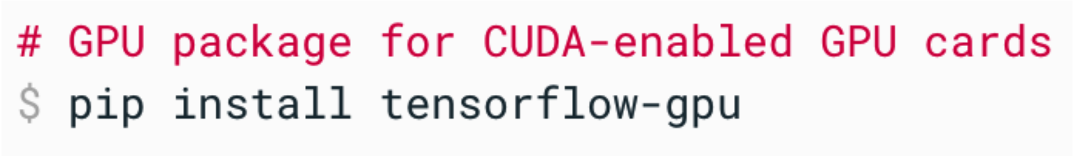
TensorFlow 1.11(stable)
python 3.6.7 (not 3.7.x)

- CPU-only

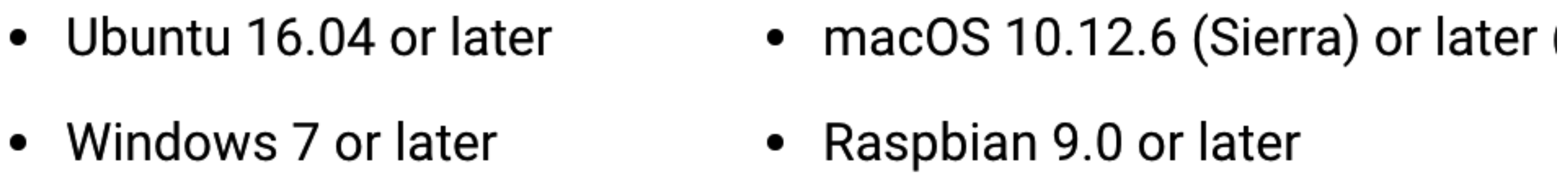


```
# Current release for CPU-only
$ pip install tensorflow
```

- GPU (nVIDIA using CUDA)

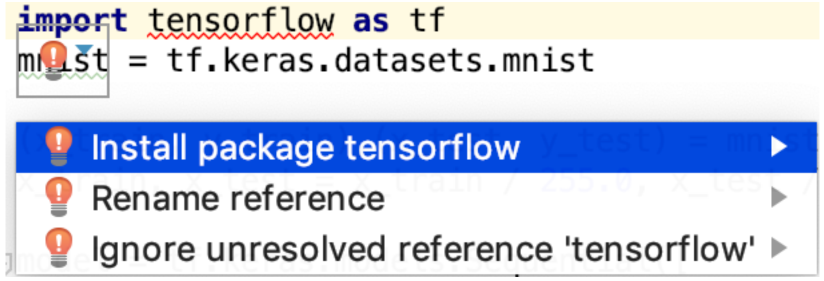


```
# GPU package for CUDA-enabled GPU cards
$ pip install tensorflow-gpu
```



- Ubuntu 16.04 or later
- Windows 7 or later
- macOS 10.12.6 (Sierra) or later
- Raspbian 9.0 or later

- Pycharm can install packages for you =>

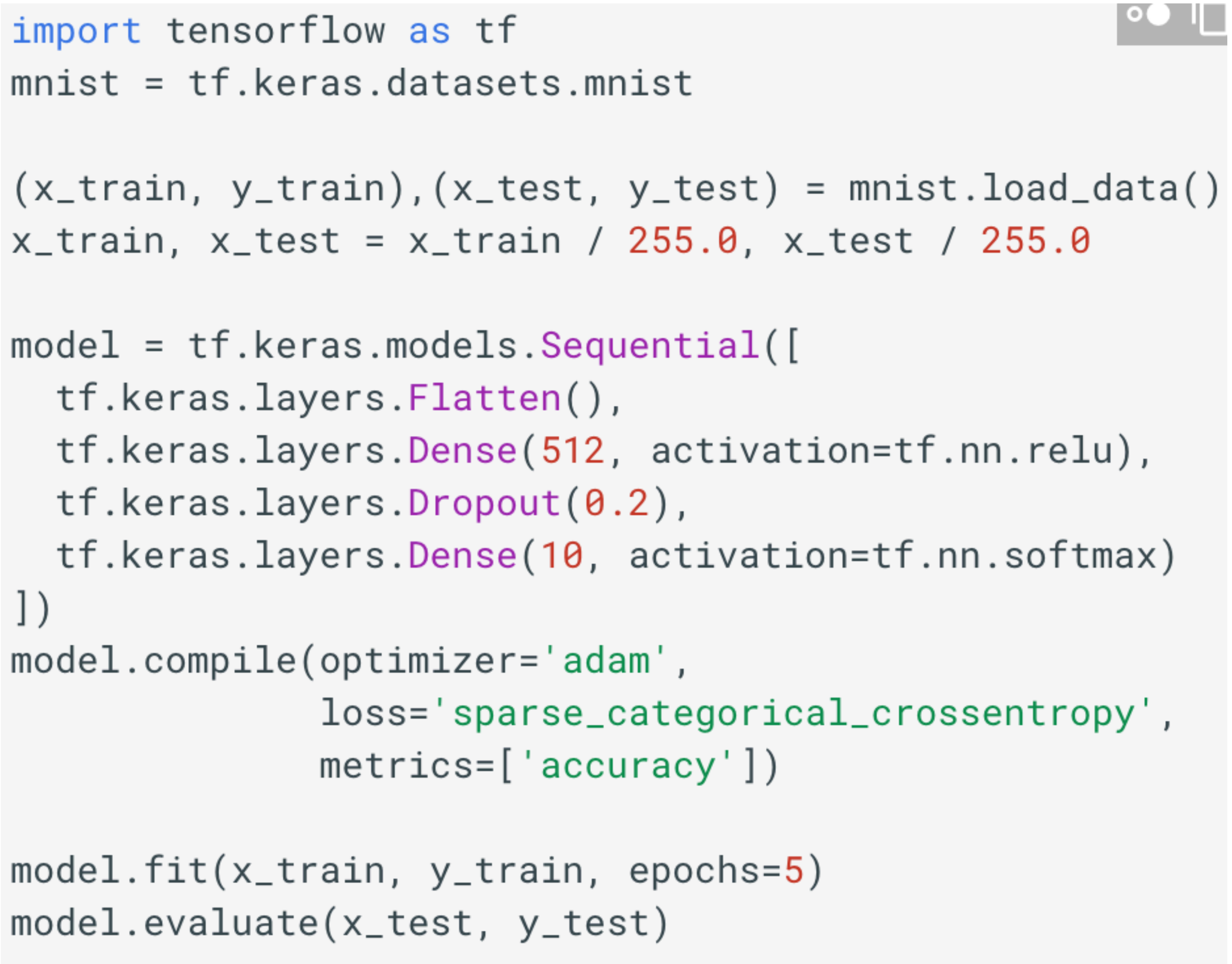


```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
```

Install package tensorflow
Rename reference
Ignore unresolved reference 'tensorflow'

Learning a pattern

1. Load the train data
2. Setup the layers
3. Compile the model
4. Train the model (fitting)
5. Evaluate the model



```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Loading Data

Using TensorFlow API => `tf.data` , `tf.placeholder`

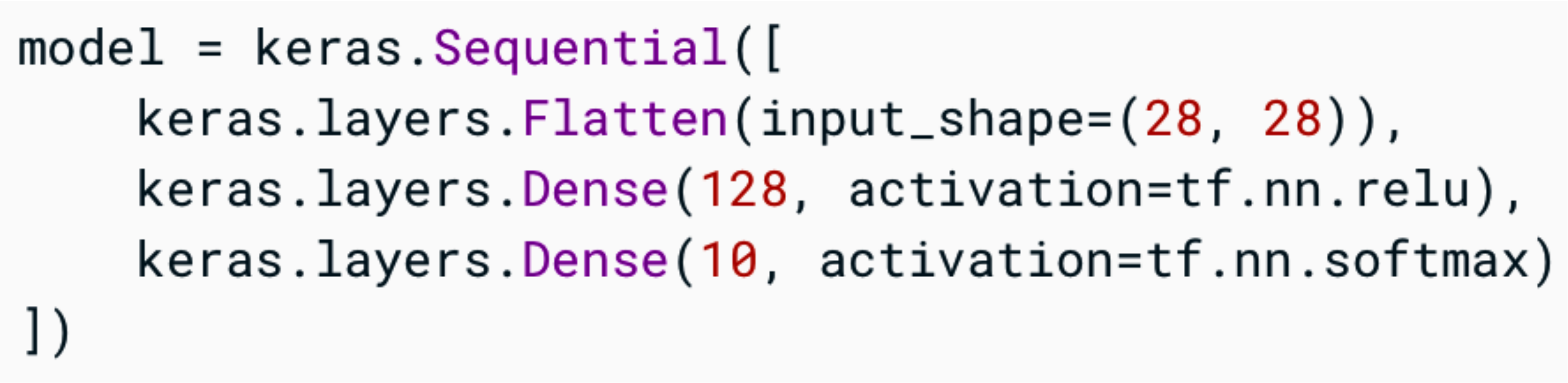
- `tf.data.Dataset`
- `tf.data.Iterator`

TensorFlow Data Pipeline (ETL):

- **Extract**: Read data from persistent storage
- **Transform**: Use CPU cores to parse and perform preprocessing operations on the data
- **Load**: Load the transformed data onto the accelerator device(s)

Model design

- Setup the layers (`tf.layers` , `keras.layers`) (e.g. Sequential model)



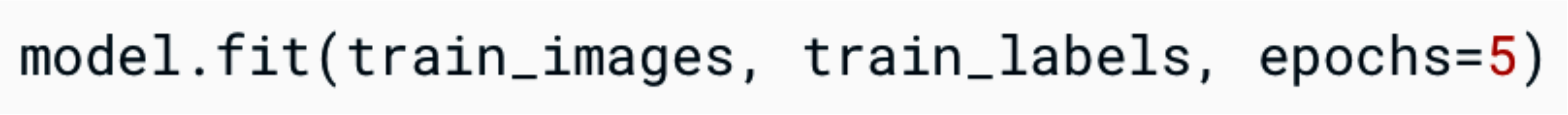
```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

- Compile the model with:
  - Loss Function (e.g. *cross-entropy*)
  - Optimizer (e.g. *gradient descent*)
  - Metrics (e.g. *accuracy*)

Model Fitting

Train phase (go and get some popcorn 🍿)

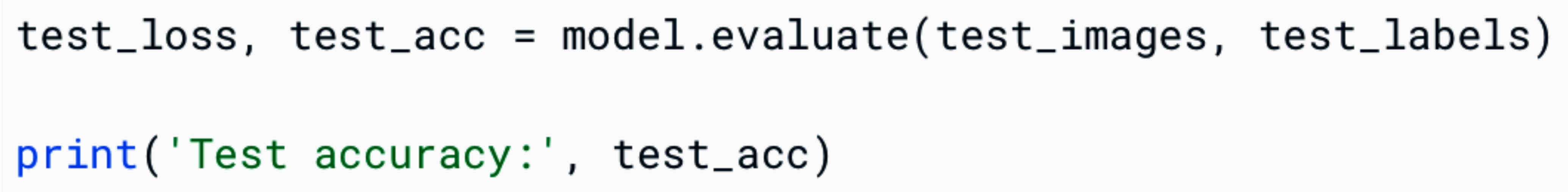
- Run the TensorFlow session (`tf.session`)
  - Use to have graphs
  - Best to have the shiny TensorBoard
  - Pure TensorFlow
- or call `model.fit()`



```
model.fit(train_images, train_labels, epochs=5)
```

Evaluation

- Evaluate model by the metrics
- Use test data with labels
- May want to use k-fold cross-validation method or so
- Check for model overfitting or underfitting

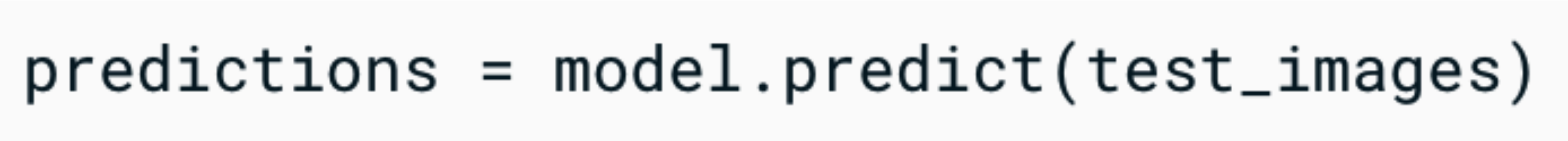


```
test_loss, test_acc = model.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)
```

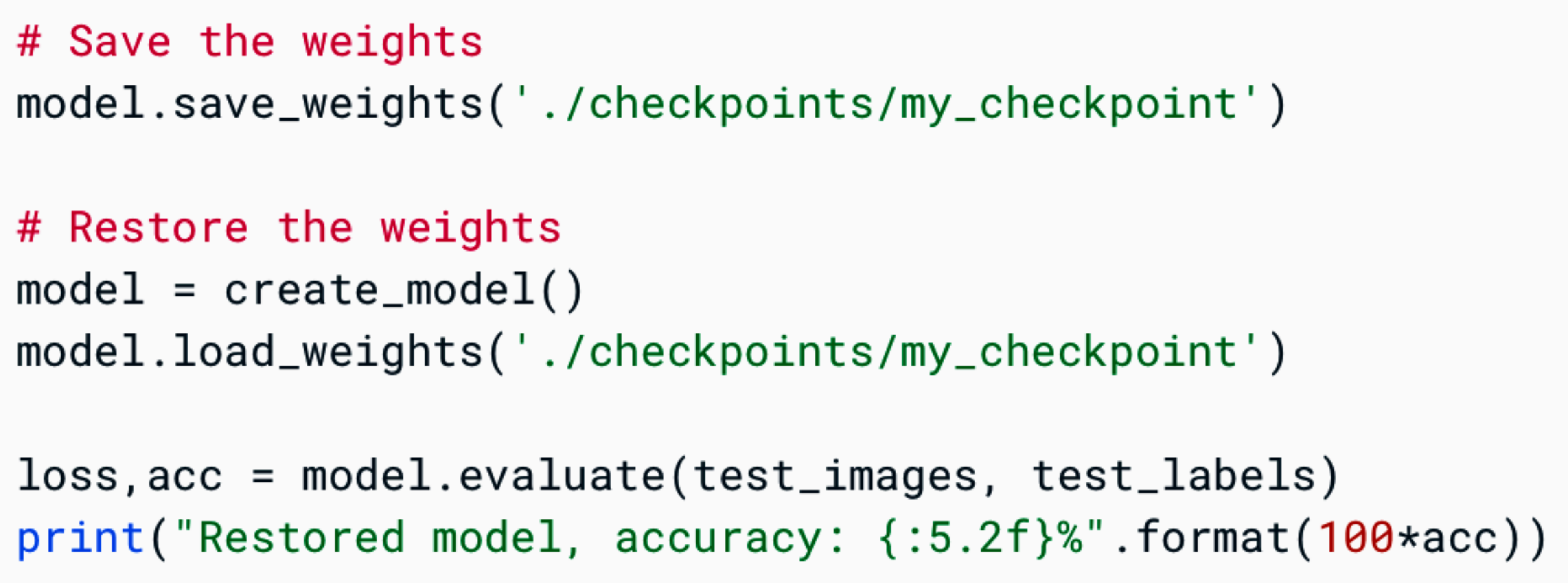
Prediction

- Use the trained/saved model to predict new data labels
- Test data has no labels
- The model will predict the labels for you
- Well done, looks like your AI is ready 😎 🥳



```
predictions = model.predict(test_images)
```

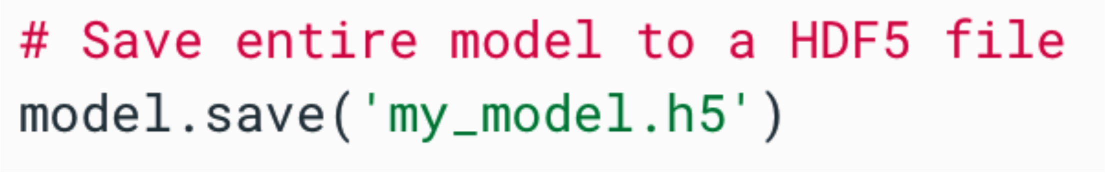
Save/Restore model



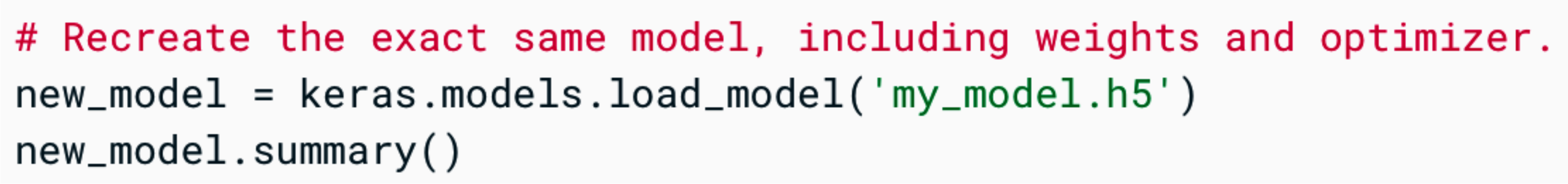
```
# Save the weights
model.save_weights('./checkpoints/my_checkpoint')

# Restore the weights
model = create_model()
model.load_weights('./checkpoints/my_checkpoint')

loss,acc = model.evaluate(test_images, test_labels)
print("Restored model, accuracy: {:5.2f}%".format(100*acc))
```



```
# Save entire model to a HDF5 file
model.save('my_model.h5')
```



```
# Recreate the exact same model, including weights and optimizer.
new_model = keras.models.load_model('my_model.h5')
new_model.summary()
```