Save, Load and Export Models in Keras

Task 1: Import Libraries ¶

In [2]:
```
import tensorflow as tf
import numpy as np
import os

print('This notebook works with TensorFlow version:', tf.__version__)

folders = ['tmp', 'models', 'model_name', 'weights']
for folder in folders:
    if not os.path.isdir(folder):
        os.mkdir(folder)

print(os.listdir('.'))
```

```
This notebook works with TensorFlow version: 2.0.1
['.ipynb_checkpoints', 'models', 'model_name', 'Student Notebook.ipynb', 'tmp', 'weights']
```

Task 2: Create Model

In [3]:
```
def create_model():
    model= tf.keras.models.Sequential([
        tf.keras.layers.Dense(128, activation='relu', input_shape=(784,)),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(10, activation='softmax')
    ])
    model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['acc'])
    return model

model= create_model()
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
_________________________________________________________________
```

Task 3: Data Preprocessing

In [4]:
```
(X_train, y_train), (X_test, y_test)= tf.keras.datasets.fashion_mnist.load_data()

X_train= np.reshape(X_train, (X_train.shape[0], 784)) / 255.
X_test= np.reshape(X_test, (X_test.shape[0], 784)) / 255.

y_train= tf.keras.utils.to_categorical(y_train)
y_test= tf.keras.utils.to_categorical(y_test) #one hot encoding
```

Task 4: Model Checkpoint During Training

In [5]:
```
checkpoint_dir= 'weights/'

_ = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=2,
              batch_size=512,
               callbacks=[tf.keras.callbacks.ModelCheckpoint(
               os.path.join(checkpoint_dir, 'epoch_{epoch:02d}_acc_{val_acc:.4f}')
               monitor='val_acc', save_weights_only= True, save_best_only= True
                                                                  )
                         ]
              )
#epoch_epoch np. save best only refers to val_acc. It will only save the weights
#of that particular checkpoint is better than the previous val accuracies.
#save weights only only saves weights but not model architecture.
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/2
60000/60000 [==============================] - 3s 52us/sample - loss: 0.7009 - acc: 0.7633 - val_loss:
Epoch 2/2
60000/60000 [==============================] - 3s 43us/sample - loss: 0.4317 - acc: 0.8479 - val_loss:
```

In [6]:
```
os.listdir(checkpoint_dir)
```

```
['checkpoint',
 'epoch_01_acc_0.8235.data-00000-of-00001',
 'epoch_01_acc_0.8235.index',
 'epoch_01_acc_0.8240.data-00000-of-00001',
 'epoch_01_acc_0.8240.index',
 'epoch_02_acc_0.8313.data-00000-of-00001',
 'epoch_02_acc_0.8313.index',
 'epoch_02_acc_0.8412.data-00000-of-00001',
 'epoch_02_acc_0.8412.index']
```

Task 5: Load Weights

In [7]:
```
model= create_model()
print(model.evaluate(X_test, y_test, verbose=False))#prints loss, accuracy
```

```
[2.33001743850708, 0.1949]
```

In [8]:
```
model.load_weights('weights/epoch_02_acc_0.8313')
print(model.evaluate(X_test, y_test, verbose=False))
```

```
[0.4669719466686249, 0.8313]
```

Task 6: Saving Complete Model During Training

In [9]:
```
models_dir= 'models/'

model=create_model()

_ = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=2,
              batch_size=512,
               callbacks=[tf.keras.callbacks.ModelCheckpoint(
               os.path.join(models_dir, 'epoch_{epoch:02d}_acc_{val_acc:.4f}.h5'),
               monitor='val_acc', save_weights_only= False, save_best_only= False
                                                              )
                          ]
              )
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/2


WARNING: Logging before flag parsing goes to stderr.
W1125 10:10:41.593647  4060 util.py:144] Unresolved object in checkpoint: (root).optimizer.iter
W1125 10:10:41.595641  4060 util.py:144] Unresolved object in checkpoint: (root).optimizer.beta_1
W1125 10:10:41.597639  4060 util.py:144] Unresolved object in checkpoint: (root).optimizer.beta_2
W1125 10:10:41.598640  4060 util.py:144] Unresolved object in checkpoint: (root).optimizer.decay
W1125 10:10:41.599642  4060 util.py:144] Unresolved object in checkpoint: (root).optimizer.learning_rat
W1125 10:10:41.600641  4060 util.py:152] A checkpoint was restored (e.g. tf.train.Checkpoint.restore or
ot all checkpointed values were used. See above for specific issues. Use expect_partial() on the load s
int.restore(...).expect_partial(), to silence these warnings, or use assert_consumed() to make the chec
rg/alpha/guide/checkpoints#loading_mechanics (https://www.tensorflow.org/alpha/guide/checkpoints#loading_mechanics) for 


60000/60000 [==============================] - 3s 47us/sample - loss: 0.6971 - acc: 0.7665 - val_loss: 
Epoch 2/2
60000/60000 [==============================] - 2s 34us/sample - loss: 0.4350 - acc: 0.8478 - val_loss: 
```

In [10]:
```
os.listdir(models_dir)
```

```
['epoch_01_acc_0.8263.h5',
 'epoch_01_acc_0.8279.h5',
 'epoch_02_acc_0.8420.h5',
 'epoch_02_acc_0.8475.h5']
```

Task 7: Load Models

In [11]:
```
model= create_model()
print(model.evaluate(X_test, y_test, verbose=False))
```

```
[2.3657102710723876, 0.0916]
```

In [13]:
```
model= tf.keras.models.load_model('models/epoch_02_acc_0.8475.h5')
print(model.evaluate(X_test, y_test, verbose=False))
model.summary()
```

```
[0.43623250589370727, 0.8475]
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_7 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_8 (Dense)              (None, 10)                1290
=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
_________________________________________________________________
```

Task 8: Manually Saving Weights and Models

In [14]:
```
model.save_weights('tmp/manually_saved.w')
os.listdir('tmp')
```

```
['checkpoint',
 'manually_saved.w.data-00000-of-00001',
 'manually_saved.w.index']
```

In [15]:
```
model.save('tmp/manually_saved.h5')#save entire model with architecture and weigh
os.listdir('tmp')
```

```
['checkpoint',
 'manually_saved.h5',
 'manually_saved.w.data-00000-of-00001',
 'manually_saved.w.index']
```

Task 9: Exporting and Restoring SavedModel Fo

In [17]:
```
model.save('model_name')#exports model to given directory in saved model format
os.listdir('model_name/')
```

```
['assets', 'saved_model.pb', 'variables']
```

In [18]:
```
model= tf.keras.models.load_model('model_name') #dir name
print(model.evaluate(X_test, y_test, verbose=False))
```

```
[0.4362325005054474, 0.8475]
```