18bce244

Tirth Hihoriya

Introduction to Keras

GitHub repo: https://github.com/keras-team/keras

What is Keras?

Keras is a deep learning framework for Python that provides a convenient way to define and train almost any kind of deep learning model. Keras is a high-level neural networks API, written in Python which is capable of running on top of **Tensorflow**, **Theano** and **CNTK**. It was developed for enabling fast experimentation.

TensorFlow Integration

Keras was originally created by *François Chollet*. Historically, Keras was a high-level API that sat on top of one of three lower-level neural network APIs and acted as a wrapper to these lower-level libraries. These libraries were referred to as Keras backend engines.

Ultimately, TensorFlow became the **most popular** backend engine for Keras. Later, Keras became integrated with the TensorFlow library and now comes completely packaged with it. Now, when you install TensorFlow, you also automatically get Keras, as it is now part of the TensorFlow library.

Differences In Imports

From a usability standpoint, many changes between the older way of using Keras with a configured backend versus the new way of having Keras integrated with TensorFlow is in the import statements.

For example, previously, we could access the `Dense` module from Keras with the following import statement.

```
>>> from keras.layers import Dense
```

Now, using Keras with TensorFlow, the import statement looks like this:

```
>>> from tensorflow.keras.layers import Dense
```

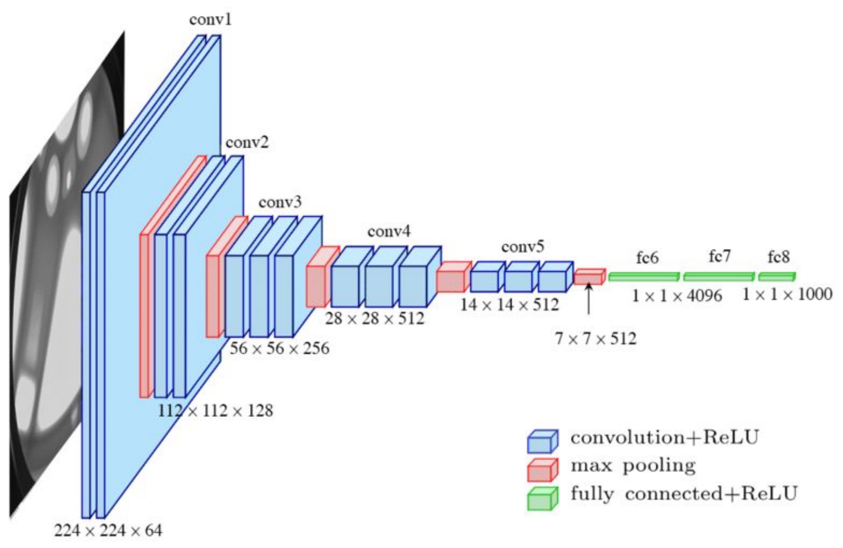
Below, you can see the difference between the old way and new way of importing some common Keras modules.

Keras Models

1) Sequential Model

A `Sequential` model is appropriate for **a plain stack of layers** where each layer has **exactly one input tensor and one output tensor**.

- Linear stack of layers
- Useful for building simple models like simple classification network and Encoder-Decoder models.
- Treat each layer as object that feeds into the next.



Sample Code:

```
import keras
from keras import layers

model = keras. Sequential()
model.add (layers.Dense(20, activation='relu', input_shape=(10,)))
model.add (layers. Dense(20, activation='relu'))
model.add (layers.Dense(10, activation='softmax'))

model.fit(x, y, epochs=10, batch_size=32)
```

2) Functional model

The Keras functional API is a way to create models that are more flexible than the *tf.keras.Sequential* API. The functional API can handle models with non-linear topology, shared layers, and even multiple inputs or outputs.

- The main idea is that a deep learning model is usually a directed acyclic graph (DAG) of layers. So the functional API is a way to build *graphs of layers*.

Sample Code:

```
import keras
from keras import layers

inputs = keras. Input (shape=(10,))
x = layers. Dense(20, activation='relu')(x)
x = layers.Dense(20, activation='relu') (x)
outputs = layers. Dense(10, activation='softmax')

model = keras. Model(inputs, outputs)
model.fit(x, y, epochs=10, batch_size=32)
```

CNN in Keras

Sample Code lines:

```
>>> model.add(Conv2D(32, (3, 3), padding="same", activation="relu"))
>>> model.add(MaxPooling2D(pool_size=(2, 2)))

>>> model.add(Conv2D(64, (3, 3), padding="same", activation="relu"))
>>> model.add(MaxPooling2D(pool_size=(2, 2)))

>>> model.add(Conv2D(128, (3, 3), padding="same", activation="relu"))
>>> model.add(MaxPooling2D(pool_size=(2, 2)))

>>> model.add(Activation("softmax"))
```

On-Line 1 we learn a total of **32 filters**. Max pooling is then used to reduce the spatial dimensions of the output volume.

We then learn **64 filters** on **Line 3**. Again max pooling is used to reduce the spatial dimensions.

The final Conv2D layer learns **128 filters**.

Layer activation functions

- **Usage of activations:**

```
>>> model.add(layers.Dense(64, activation=activations.relu))
```

All built-in activations may also be passed via their string identifier:

```
>>> model.add(layers.Dense(64, activation='relu'))
```

- **Available activations**

➔ `relu` function
➔ `sigmoid` function
➔ `softmax` function
➔ `softplus` function
➔ `softsign` function
➔ `tanh` function
➔ `selu` function
➔ `elu` function
➔ `exponential` function

Optimizers

- **Usage of optimizers:**

An optimizer is one of the two arguments required for compiling a Keras model:

```
# pass optimizer by name: default parameters will be used
>>> model.compile(loss='categorical_crossentropy', optimizer='adam')
```

- **Available activations**

➔ SGD

- ➔ RMSprop
- ➔ Adam
- ➔ Adadelta
- ➔ Adagrad
- ➔ Adamax
- ➔ Nadam
- ➔ Ftrl

Metrics

- **Usage of metrics:**

```
>>> tf.keras.metrics.Accuracy(name="accuracy", dtype=None)
```

- ➔ Usage with `compile()` API:

```
>>> model.compile(optimizer='sgd',
                  loss='mse',
                  metrics=[tf.keras.metrics.BinaryAccuracy()])
```

- **Available metrics:**

Accuracy metrics

- ➔ Accuracy class
- ➔ BinaryAccuracy class
- ➔ CategoricalAccuracy class
- ➔ TopKCategoricalAccuracy class
- ➔ SparseTopKCategoricalAccuracy class

Probabilistic metrics

- ➔ BinaryCrossentropy class
- ➔ CategoricalCrossentropy class
- ➔ SparseCategoricalCrossentropy class
- ➔ KLDivergence class
- ➔ Poisson class

Regression metrics

- ➔ MeanSquaredError class
- ➔ RootMeanSquaredError class
- ➔ MeanAbsoluteError class
- ➔ MeanAbsolutePercentageError class
- ➔ MeanSquaredLogarithmicError class
- ➔ CosineSimilarity class
- ➔ LogCoshError class

Classification metrics based on True/False positives & negatives

- ➔ AUC class
- ➔ Precision class
- ➔ Recall class
- ➔ TruePositives class
- ➔ TrueNegatives class
- ➔ FalsePositives class
- ➔ FalseNegatives class
- ➔ PrecisionAtRecall class
- ➔ SensitivityAtSpecificity class
- ➔ SpecificityAtSensitivity class

Losses

- **Usage of losses:**

  The purpose of loss functions is to compute the quantity that a model should seek to minimize during training.

```
>>> tf.keras.losses.MeanSquaredError(reduction="auto", name="mean_squared_error")
```

Usage with the `compile()` API:

```
>>> model.compile(optimizer='sgd', loss=tf.keras.losses.MeanSquaredError())
```

- **Available activations**

Probabilistic losses

- ➔ BinaryCrossentropy class
- ➔ CategoricalCrossentropy class

- ➔ SparseCategoricalCrossentropy class
- ➔ Poisson class
- ➔ binary_crossentropy function
- ➔ categorical_crossentropy function
- ➔ sparse_categorical_crossentropy function
- ➔ poisson function
- ➔ KLDivergence class
- ➔ kl_divergence function

Regression losses

- ➔ MeanSquaredError class
- ➔ MeanAbsoluteError class
- ➔ MeanAbsolutePercentageError class
- ➔ MeanSquaredLogarithmicError class
- ➔ CosineSimilarity class
- ➔ mean_squared_error function
- ➔ mean_absolute_error function
- ➔ mean_absolute_percentage_error function
- ➔ mean_squared_logarithmic_error function
- ➔ cosine_similarity function
- ➔ Huber class
- ➔ huber function
- ➔ LogCosh class
- ➔ log_cosh function

The Keras ecosystem

The Keras project isn't limited to the core Keras API for building and training neural networks. It spans a wide range of related initiatives that cover every step of the machine learning workflow.

- **Keras Tuner**
- **AutoKeras**
- **TensorFlow Cloud**
- **TensorFlow.js**
- **TensorFlow Lite**
- **Model optimization toolkit**
- **TFX integration**