Keras

Keras is an easy-to-use library for building and training deep learning models. It provides a simple way to create complex neural networks without dealing with complicated details. Keras works with TensorFlow, which helps to run the models. You can use Keras to build different types of models, like those for image recognition or analyzing text. Its clear and straightforward design makes it a popular choice for beginners and experts who want to quickly try out new ideas in deep learning.

Installing Keras

```
pip install keras
```

Model Creation

Function	Description
Sequential()	Creates a linear stack of layers, useful for simple models.
Model()	Used to create a more flexible model by specifying inputs and outputs.

Layer Types

Function	Description
Dense()	Creates a fully connected layer, where each neuron is connected to every neuron in the previous layer.
Conv2D()	Creates a 2D convolutional layer, used for processing images.
MaxPooling2D()	Performs max pooling operation, reducing the size of the data.
LSTM()	Creates a Long Short-Term Memory (LSTM) layer, useful for sequential data like text.
Dropout()	Adds dropout regularization to prevent overfitting by randomly setting a fraction of inputs to zero.
Embedding()	Creates an embedding layer, useful for working with categorical or sequential data, such as words.
Flatten()	Flattens the input data, often used to convert the output of convolutional layers to a 1D array for fully connected layers.
GlobalAveragePooling2D()	Applies global average pooling to 2D data, reducing the size by averaging across all the spatial dimensions.
Concatenate()	Merges multiple models or layers into one layer.

Model Training

Function	Description
compile()	Configures the model for training by specifying the optimizer, loss function, and metrics.
fit()	Trains the model on the input data for a specified number of epochs.
fit_generator()	Trains the model on data generated in real time by a Python generator.
evaluate()	Evaluates the performance of the model on a dataset and returns the loss and metrics.
evaluate_generator()	Evaluates the model using a data generator, useful for large datasets that don't fit in memory.
predict()	Uses a trained model to make predictions on new data.
train_on_batch()	Trains the model on a single batch of data at a time.
test_on_batch()	Tests the model on a single batch of data.

Callbacks

Function	Description
EarlyStopping()	Stops training early when a monitored metric stops improving.
ModelCheckpoint()	Saves the model after each epoch if it has improved, useful for continuing training later.
ReduceLROnPlateau()	Reduces the learning rate when a metric has stopped improving.
TensorBoard()	Logs training metrics to visualize in TensorBoard, useful for tracking progress.

Loss Functions

Function	Description
binary_crossentropy()	Computes binary cross-entropy loss, used for binary classification tasks.
categorical_crossentropy()	Computes categorical cross-entropy loss, used for multi-class classification tasks.
mean_squared_error()	Computes the mean squared error loss, used for regression tasks.

Metrics

Function	Description
accuracy()	Computes the accuracy of the model, often used for classification tasks.
AUC()	Computes the area under the ROC curve, used for binary classification tasks.

Keras Cheat-Sheet

Keras is an easy-to-use library for building and training neural networks. It helps create complex models for tasks like image recognition, language processing, and more. Keras provides simple ways to design models using two main types of tools: Sequential (for basic models) and Functional (for more complex ones). It works well with TensorFlow, which helps run the models efficiently. Keras is popular because it's simple to learn yet powerful enough for experts to use for serious deep-learning projects.

Importing Keras

```
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D, LSTM
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Model Creation

Sequential Model

```
model = Sequential()
```

Functional Model

```
input = Input(shape=(input_shape))
x = Dense(64)(input)
output = Dense(1)(x)
model = Model(inputs=input, outputs=output)
```

Adding Layers

Dense Layer

```
model.add(Dense(units=64, activation='relu', input_shape=(input_size,)))
```

Convolutional Layer (Conv2D)

```
model.add(Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(height, width, channels)))
```

Max Pooling Layer (MaxPooling2D)

```
model.add(MaxPooling2D(pool_size=(2, 2)))
```

Flatten Layer

```
model.add(Flatten())
```

Model Compilation

```
model.compile(optimizer=Adam(),
loss='binary_crossentropy', metrics=['accuracy'])
```

Model Training

```
history = model.fit(X_train, y_train, epochs=10,
batch_size=32, validation_data=(X_val, y_val))
```

Custom Layer

```
from tensorflow.keras import layers

class CustomLayer(layers.Layer):
  def __init__(self):
    super(CustomLayer, self).__init__()

  def call(self, inputs):
    return inputs * 2
```

Saving & Loading Model Weights

Saving Model Weight

```
model.save_weights('model_weights.h5')
```

Loading Model Weight

```
model.load_weights('model_weights.h5')
```

Model Evaluation

```
loss, accuracy = model.evaluate(X_test, y_test)
```

Model Prediction

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

Callbacks

Early Stopping

```
early_stop = EarlyStopping(monitor='val_loss',
patience=5)
```

Data Augmentation

ImageDataGenerator

```
datagen = ImageDataGenerator(rotation_range=40,
width_shift_range=0.2, height_shift_range=0.2)
datagen.fit(X_train)
```

Optimizers

Adam Optimizer

```
optimizer = Adam(learning_rate=0.001)
```

SGD Optimizer

```
optimizer = SGD(learning_rate=0.01, momentum=0.9)
```

Saving and Loading Models

Save Mode

```
model.save('model.h5')
```

Load Mode

```
from tensorflow.keras.models import load_model
model = load_model('model.h5')
```

Custom Loss Function

```
import tensorflow as tf

def custom_loss(y_true, y_pred):
  return tf.reduce_mean(tf.square(y_true - y_pred))
```

GeeksforGeeks