Keras Cheat Sheet

A concise cheat sheet for Keras, covering fundamental concepts, common layers, model building, training, and evaluation techniques for deep learning.

Core Concepts

Sequential Model

A linear stack of layers.
Useful for simple, feed-forward networks.

```
model = keras.Sequential([
    layers.Dense(64, activation='relu',
input_shape=(input_dim,)),
    layers.Dense(10, activation='softmax')
])
```

Adding Layers:

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

Functional API

A more flexible way to define models as graphs of layers. Allows for complex architectures like multi-input/output models, shared layers, etc.

```
inputs = keras.Input(shape=(input_dim,))
x = layers.Dense(64, activation='relu')
(inputs)
outputs = layers.Dense(10,
activation='softmax')(x)
model = keras.Model(inputs=inputs,
outputs=outputs)
```

Layers

`Dense`	Fully connected layer.
`Conv2D`	2D convolutional layer (for images).
`MaxPooling2D`	Max pooling layer.
`LSTM`	Long Short-Term Memory layer (for sequences).
`Embedding`	Embedding layer (for representing words as vectors).

Model Building

Defining the Model

Using the Sequential API:

```
model = keras.Sequential([
    layers.Dense(128, activation='relu',
input_shape=(784,)),
    layers.Dropout(0.3),
    layers.Dense(10, activation='softmax')
])
```

Using the Functional API:

```
input_layer = keras.Input(shape=(784,))
x = layers.Dense(128, activation='relu')
(input_layer)
x = layers.Dropout(0.3)(x)
output_layer = layers.Dense(10,
activation='softmax')(x)
model = keras.Model(inputs=input_layer,
outputs=output_layer)
```

Compiling the Model

Specifying the optimizer, loss function, and metrics.

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Optimizers: `adam`, `rmsprop`, `sgd`
Loss Functions: `categorical_crossentropy`, `binary_crossentropy`, `mse`
Metrics: `accuracy`, `precision`, `recall`

Common Layers

`Dense(units, activation='relu')`	A fully-connected layer with ReLU activation.
`Conv2D(filters, kernel_size, activation='relu')`	2D convolutional layer for image processing.
`MaxPooling2D(pool_size=(2, 2))`	Max pooling layer to reduce spatial dimensions.
`Dropout(rate)`	Dropout layer to prevent overfitting.

Training and Evaluation

Training the Model

Training the model on the training data.

```
model.fit(x_train, y_train, epochs=10,
batch_size=32)
```

Parameters:

`epochs`: Number of training iterations over the entire dataset.

`batch_size`: Number of samples per gradient update.

Callbacks:

Used to customize the training process. Examples:

- `ModelCheckpoint`: Save the best model during training.
- `EarlyStopping`: Stop training when a metric has stopped improving.

```
callbacks = [

keras.callbacks.ModelCheckpoint(filepath='mode
l.h5', save_best_only=True),

keras.callbacks.EarlyStopping(monitor='val_los
s', patience=3)
]
model.fit(x_train, y_train, epochs=10,
batch_size=32, validation_data=(x_val, y_val),
callbacks=callbacks)
```

Evaluating the Model

Evaluating the model on the test data.

```
loss, accuracy = model.evaluate(x_test,
y_test)
print('Test accuracy:', accuracy)
```

Prediction

Making predictions with the model.

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

Advanced Features

Regularization

L1 Regularization	Adds a penalty equal to the absolute value of the magnitude of coefficients. `keras.regularizers.L1(0.01)`
L2 Regularization	Adds a penalty equal to the square of the magnitude of coefficients. `keras.regularizers.L2(0.01)`
Elastic Net Regularization	A combination of L1 and L2 regularization. `keras.regularizers.ElasticNet(l1=0.01, l2=0.01)`

Batch Normalization

Normalizes the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.

```
layers.BatchNormalization()
```

Saving and Loading Models

Saving the model:

```
model.save('my_model.keras')
```

Loading the model:

```
model =
keras.models.load_model('my_model.keras')
```