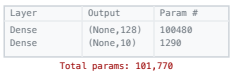
Keras

The essentials of Keras, the high-level way to build and train neural networks, from stacking layers to callbacks and transfer learning.

Build a Model

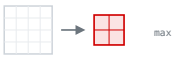
Three ways to assemble layers into a model.

PURPOSE	COMMAND	RESULT
Linear stack of layers.	`model = keras.Sequential([...])`	
Declare the input shape.	`keras.Input(shape=(784,))`	
One dense hidden layer.	`keras.layers.Dense(128, "relu")`	
Functional API (a graph).	`outputs = keras.layers.Dense(10)(x)`	
Subclass for full control.	`class Net(keras.Model): def call(self, x):`	
Inspect the architecture.	`model.summary()`	

Sequential is the quick stack; the functional API handles branches; subclassing gives a custom call

The Layer Catalog

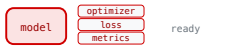
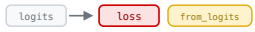
The building blocks you wire together.

PURPOSE	COMMAND	RESULT
Fully connected layer.	`keras.layers.Dense(64, "relu")`	
2D convolution (images).	`keras.layers.Conv2D(32, 3, "relu")`	
Downsample feature maps.	`keras.layers.MaxPooling2D(2)`	
Flatten / regularize.	`keras.layers.Flatten()` `Dropout(0.5)`	
Sequence layers.	`Embedding(10000, 64)` `LSTM(64)`	
Scale inputs in-graph.	`keras.layers.Rescaling(1./255)`	

Dense for tabular and heads; Conv2D/MaxPooling for images; Embedding/LSTM for sequences

Compile the Model

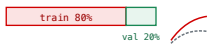
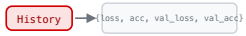
Choose the optimizer, loss, and metrics before training.

PURPOSE	COMMAND	RESULT
Wire up training.	`model.compile(optimizer, loss, metrics)`	
Pick an optimizer + LR.	`keras.optimizers.Adam( learning_rate=1e-3)`	
Multiclass loss (int labels).	`loss="sparse_categorical_crossentropy"`	
Binary / regression loss.	`"binary_crossentropy""mse"`	
Track extra metrics.	`metrics=["accuracy",keras.metrics.AUC()]`	
Object form (fine control).	`keras.losses.SparseCategoricalCrossentropy( from_logits=True)`	

match the loss to the labels: sparse_categorical for int multiclass, binary_crossentropy for two-class, mse for regression

Train, Evaluate, Predict

The fit / evaluate / predict loop over your data.

PURPOSE	COMMAND	RESULT
Train on arrays.	`model.fit(x, y, epochs=10, batch_size=32)`	
Hold out a validation set.	`model.fit(..., validation_split=0.2)`	
Capture the history.	`history = model.fit(...)`	
Score on test data.	`model.evaluate(x_test, y_test)`	
Predict new examples.	`preds = model.predict(x_new)`	
Feed a tf.data pipeline.	`model.fit(train_ds, epochs=10)`	

fit returns a History of per-epoch metrics; all three accept NumPy arrays or streamed tf.data datasets

Callbacks

Hooks that watch and steer training between epochs.

PURPOSE	COMMAND	RESULT
Stop when val stops improving.	`EarlyStopping(patience=3, restore_best_weights=True)`	
Save the best model so far.	`ModelCheckpoint("best.keras", save_best_only=True)`	
Drop LR on a plateau.	`ReduceLROnPlateau(factor=0.5, patience=2)`	
Log to TensorBoard.	`TensorBoard(log_dir="logs")`	
Append metrics to CSV.	`CSVLogger("training.csv")`	
Pass the list to fit.	`model.fit(..., callbacks=[es, ckpt, rlrop])`	

collect callbacks in a list and hand it to fit(..., callbacks=[...]); they fire at batch and epoch edges

Save & Load

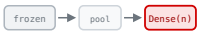
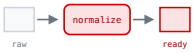
Persist a model, its weights, or its config to disk.

PURPOSE	COMMAND	RESULT
Save the whole model.	`model.save("model.keras")`	
Load it back, ready to use.	`keras.models.load_model("model.keras")`	
Save weights only.	`model.save_weights("m.weights.h5")`	
Restore weights into a model.	`model.load_weights("m.weights.h5")`	
Save architecture as JSON.	`json_str = model.to_json()`	
Export for serving.	`model.export("served_model")`	

.keras captures arch + weights + compile; save_weights stores only the numbers; export writes a SavedModel

Transfer Learning

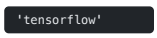
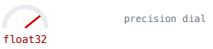
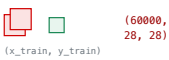
Reuse a pretrained network and fine-tune a small head.

PURPOSE	COMMAND	RESULT
Load a pretrained backbone.	`base = keras.applications.MobileNetV2(...)`	
Freeze the backbone.	`base.trainable =False`	
Add your own head.	`GlobalAveragePooling2D() +Dense`	
Preprocess to match the model.	`mobilenet_v2.preprocess_input(x)`	
Train just the head.	`model.compile(...); model.fit(epochs=5)`	
Unfreeze + fine-tune low LR.	`base.trainable =TrueAdam(1e-5)`	

freeze the backbone and train the head, then optionally unfreeze and fine-tune at a tiny learning rate

Backends & Config

One Keras 3 API, your choice of TensorFlow, JAX, or PyTorch.

PURPOSE	COMMAND	RESULT
Pick the backend (before import).	`os.environ["KERAS_BACKEND"] = "jax"`	
Confirm the active backend.	`keras.backend.backend()`	
Backend-agnostic tensor ops.	`keras.ops.matmul(a, b) ops.relu(x)`	
Make a run reproducible.	`keras.utils.set_random_seed(42)`	
Set default float precision.	`keras.config.set_floatx("float32")`	
Built-in datasets to start.	`keras.datasets.mnist.load_data()`	

set KERAS_BACKEND before importing keras; write once against keras.ops; reach for keras.datasets to experiment