#_ important Keras Operations [+100]

Model Building and Architecture:

- keras.models.Sequential(): Linear stack of layers.
- keras.Model(inputs, outputs): Model class used to create a functional API model.
- keras.layers.Dense(units, activation): Regular densely-connected NN layer.
- keras.layers.Conv2D(filters, kernel_size): 2D convolution layer.
- keras.layers.MaxPooling2D(pool_size): Max pooling operation for 2D spatial data.
- keras.layers.AveragePooling2D(pool_size): Average pooling for 2D spatial data.
- keras.layers.GlobalMaxPooling2D(): Global max pooling operation for 2D spatial data.
- keras.layers.GlobalAveragePooling2D(): Global average pooling operation for 2D spatial data.
- keras.layers.Flatten(): Flattens the input without affecting batch size.
- keras.layers.Dropout(rate): Applies Dropout to the input.
- keras.layers.BatchNormalization(): Batch normalization layer.
- keras.layers.Activation(activation): Applies an activation function to an output.
- keras.layers.Embedding(input_dim, output_dim): Turns positive integers into dense vectors of fixed size.

Recurrent Layers:

- keras.layers.LSTM(units): Long Short-Term Memory layer.
- keras.layers.GRU(units): Gated Recurrent Unit layer.
- keras.layers.SimpleRNN(units): Fully-connected RNN where the output is fed back to the input.

Functional API:

By: Waleed Mousa

- keras.Input(shape): Used to instantiate a Keras tensor.
- keras.Model(inputs, outputs): Create a model from input and output tensors.

Compiling a Model:

- model.compile(optimizer, loss, metrics): Configures the model for training.
- keras.optimizers.Adam(learning_rate): Adam optimizer.
- keras.optimizers.SGD(learning_rate): Stochastic gradient descent optimizer.
- keras.optimizers.RMSprop(learning_rate): RMSprop optimizer.
- keras.losses.CategoricalCrossentropy(): Compute the crossentropy loss between labels and predictions.
- keras.losses.BinaryCrossentropy(): Compute the crossentropy loss between true labels and predicted labels.
- keras.losses.MeanSquaredError(): Computes the mean of squares of errors between labels and predictions.
- keras.metrics.Accuracy(): Calculates how often predictions match binary labels.
- keras.metrics.Precision(): Computes the precision of the predictions with respect to the labels.
- keras.metrics.Recall(): Computes the recall of the predictions with respect to the labels.

Training and Evaluation:

- model.fit(x, y, epochs, batch_size): Trains the model for a fixed number of epochs.
- model.evaluate(x, y, batch_size): Returns the loss value and metrics values for the model.
- model.predict(x, batch_size): Generates output predictions for the input samples.
- model.summary(): Prints a string summary of the network.

Saving and Loading Models:

- `model.save(filepath)`: Saves a Keras model as an HDF5 file.
- `keras.models.load_model(filepath)`: Loads a model saved via `model.save`.
- `model.save_weights(filepath)`: Saves the weights of the model.
- `model.load_weights(filepath)`: Loads the weights of the model.

Callbacks:

- `keras.callbacks.ModelCheckpoint(filepath)`: Save the model after every epoch.
- `keras.callbacks.EarlyStopping(monitor, patience)`: Stop training when a monitored metric has stopped improving.
- `keras.callbacks.TensorBoard(log_dir)`: Enable visualizations for TensorBoard.
- `keras.callbacks.ReduceLROnPlateau()`: Reduce learning rate when a metric has stopped improving.
- `keras.callbacks.LearningRateScheduler(schedule)`: Learning rate scheduler.

Data Preprocessing:

- `keras.preprocessing.image.ImageDataGenerator()`: Generate batches of tensor image data with real-time data augmentation.
- `keras.preprocessing.sequence.pad_sequences(sequences)`: Pads sequences to the same length.
- `keras.preprocessing.text.Tokenizer(num_words)`: Text tokenization utility class.

Advanced Layers:

- `keras.layers.Conv2DTranspose(filters, kernel_size)`: Transposed convolution layer.
- `keras.layers.Reshape(target_shape)`: Reshapes an output to a certain shape.
- `keras.layers.Concatenate()`: Layer that concatenates a list of inputs.
- `keras.layers.Add()`: Layer that adds a list of inputs.

By: Waleed Mousa

- keras.layers.Multiply(): Layer that multiplies a list of inputs.
- keras.layers.UpSampling2D(size): Upsampling layer for 2D inputs.
- keras.layers.LeakyReLU(alpha): Leaky version of a Rectified Linear Unit.
- keras.layers.PReLU(): Parametric Rectified Linear Unit.
- keras.layers.ELU(alpha): Exponential Linear Unit.

Custom Layers and Models:

- class MyLayer(keras.layers.Layer): Create a custom layer.
- class MyModel(keras.Model): Create a custom model.
- def call(self, inputs): Specify the logic of the layer/model.
- def get_config(self): Save configuration for the layer/model.

Image and **Sequence** Utilities:

- keras.utils.to_categorical(y, num_classes): Converts a class vector to binary class matrix.
- keras.utils.plot_model(model, to_file): Convert a Keras model to dot format and plot.

Optimization and Regularization Techniques:

- keras.regularizers.l1(l1=0.01): Apply L1 regularization.
- keras.regularizers.l2(l2=0.01): Apply L2 regularization.
- keras.regularizers.l1_l2(l1=0.01, l2=0.01): Apply L1-L2 regularization.

Custom Loss Functions **and** Metrics:

- def custom_loss_function(y_true, y_pred): Define a custom loss function.
- def custom_metric_function(y_true, y_pred): Define a custom metric function.
- model.compile(loss=custom_loss_function, metrics=[custom_metric_function]): Use custom loss and metrics.

Advanced **Model** Functionalities:

- `model.layers[index]`: Access a layer by its index.
- `model.get_layer(name)`: Access a layer by its name.
- `model.trainable = False`: Freeze the layers of the model.
- `model.get_weights()`: Returns the weights of the model.
- `model.set_weights(weights)`: Sets the weights of the model.
- `keras.layers.InputLayer(input_shape)`: Add an input layer to the model.
- **TensorBoard Integration**:
- `keras.callbacks.TensorBoard(log_dir)`: Log data for visualization with TensorBoard.
- `tensorboard --logdir=path_to_your_logs`: Start TensorBoard to visualize metrics.

Functional API for Complex Models:

- `input = keras.layers.Input(shape)`: Define an input layer for functional API.
- `output = keras.layers.Dense(units)(input)`: Connect layers in a functional API.
- `model = keras.Model(inputs, outputs)`: Create a model with multiple inputs/outputs.

Model Visualization:

- `keras.utils.plot_model(model, to_file='model.png')`: Plot the model architecture.
- `keras.utils.model_to_dot(model)`: Convert the model to dot format.

Gradient Manipulation and Custom Training:

- `with tf.GradientTape() as tape`: Record operations for automatic differentiation.
- `gradients = tape.gradient(loss, model.trainable_variables)`: Compute gradients.
- `optimizer.apply_gradients(zip(gradients, model.trainable_variables))`: Apply gradients to variables.

By: Waleed Mousa

Keras Applications (Pre-trained **Models**):

- `keras.applications.ResNet50(weights='imagenet')`: Pre-trained ResNet50 model.
- `keras.applications.VGG16(weights='imagenet')`: Pre-trained VGG16 model.
- `keras.applications.MobileNetV2(weights='imagenet')`: Pre-trained MobileNetV2 model.

Advanced Training Techniques:

- `model.fit_generator(generator, epochs, steps_per_epoch)`: Train a model on data yielded batch-by-batch by a generator.
- `model.train_on_batch(x, y)`: Run a single gradient update on a single batch of data.

Hyperparameter Tuning:

- `keras.wrappers.scikit_learn.KerasClassifier(build_fn)`: Implementation of the scikit-learn classifier API for Keras.
- `keras.wrappers.scikit_learn.KerasRegressor(build_fn)`: Implementation of the scikit-learn regressor API for Keras.

Sequence and Time Series:

- `keras.preprocessing.sequence.TimeseriesGenerator(data, targets, length)`: Utility class for generating batches of temporal data.
- `keras.layers.LSTM(units)`: Long Short-Term Memory layer for sequence data.
- `keras.layers.Bidirectional(layer)`: Bidirectional wrapper for RNNs.

Advanced Text Processing:

- `keras.layers.Embedding(input_dim, output_dim)`: Turns positive integers (indexes) into dense vectors of fixed size.
- `keras.preprocessing.text.Tokenizer(num_words)`: Text tokenization utility class.

By: Waleed Mousa

- keras.preprocessing.text.one_hot(text, n): One-hot encodes a text into a list of word indexes.
- keras.preprocessing.sequence.pad_sequences(sequences, maxlen): Pads sequences to the same length.

Functional API Complex Models:

- keras.layers.concatenate(inputs): Layer that concatenates a list of inputs.
- keras.layers.add(inputs): Layer that adds a list of inputs.
- keras.layers.subtract(inputs): Layer that subtracts two inputs.
- keras.layers.multiply(inputs): Layer that multiplies a list of inputs.

Regularization and Normalization:

- keras.layers.Dropout(rate): Applies Dropout to the input to prevent overfitting.
- keras.layers.AlphaDropout(rate): Applies Alpha Dropout, a Dropout that keeps mean and variance to their original values.
- keras.layers.GaussianNoise(stddev): Apply additive zero-centered Gaussian noise.
- keras.layers.LayerNormalization(): Layer normalization layer.

Custom Layers and Models:

- class CustomLayer(keras.layers.Layer): Build a custom layer by subclassing.
- def build(self, input_shape): Create the state of the layer (weights).
- def call(self, inputs): Defines the computation from inputs to outputs.

Callbacks and Custom Callbacks:

- keras.callbacks.LambdaCallback(on_epoch_end=lambda epoch, logs: ...): Quick custom callback with lambda functions.

By: Waleed Mousa

- `class CustomCallback(keras.callbacks.Callback)`: Create a custom callback.
- `def on_epoch_end(self, epoch, logs=None)`: Method called at the end of an epoch.

Keras Utils:

- `keras.utils.get_file(fname, origin)`: Downloads a file from a URL if it not already in the cache.
- `keras.utils.to_categorical(y, num_classes)`: Converts a class vector (integers) to binary class matrix.
- `keras.utils.normalize(x, axis)`: Normalizes a Numpy array.

Metrics and Losses:

- `keras.metrics.AUC()`: Computes the approximate AUC (Area under the curve) via a Riemann sum.
- `keras.losses.Huber()`: Computes the Huber loss, a less sensitive loss to outliers than the mean square error.
- `keras.losses.LogCosh()`: Computes the logarithm of the hyperbolic cosine of the prediction error.

Advanced **Functional** API:

- `keras.layers.Input(shape)`: Used to instantiate a Keras tensor for use with the functional API.
- `model = keras.Model(inputs=input_layer, outputs=output_layer)`: Create a Keras model with multiple input and output layers.
- `keras.layers.Concatenate(axis)`: Layer that concatenates a list of inputs along a specific axis.

Model Optimization **and** Tuning:

- `keras.optimizers.Nadam(learning_rate)`: Nesterov Adam optimizer.
- `keras.optimizers.Adadelta(learning_rate)`: Adadelta optimizer.
- `keras.optimizers.Ftrl(learning_rate)`: FTRL optimizer.

Image Data Augmentation:

By: Waleed Mousa

- keras.preprocessing.image.ImageDataGenerator(rotation_range, width_shift_range): Generate batches of tensor image data with real-time data augmentation.
- datagen.flow(x, y, batch_size): Takes data & label arrays, generates batches of augmented data.

Saving and Serializing Models:

- model.save('my_model.h5'): Save a Keras model into a single HDF5 file.
- keras.models.load_model('my_model.h5'): Load a Keras model from an HDF5 file.

Advanced RNN Layers:

- keras.layers.ConvLSTM2D(filters, kernel_size): Convolutional LSTM.
- keras.layers.SimpleRNNCell(units): Cell class for SimpleRNN.
- keras.layers.GRUCell(units): Cell class for the GRU layer.

Keras **Backend:**

- keras.backend.function(inputs, outputs): Instantiates a Keras function.
- keras.backend.gradients(loss, variables): Returns the gradients of loss w.r.t. variables.

By: Waleed Mousa