365 DataScience Pooling layers with TensorFlow

```
# Importing the relevant packages
import tensorflow as tf

# The outlined code below is to show how we can add a pooling layer to a convolutional network,
# It does not include any actual data, thus, cannot be trained
# You can include any image data you want, after properly preprocessing it

# Tensorflow the process of creation of neural networks to the following steps:
# - defining a model variable with the different layers
# - compiling the model variable and specifying the optimizer and loss function
# - OPTIONAL: defining early stopping callback
# - training the model with '.fit()' method
```

Creating the model

```
# Outlining the model/architecture of our network
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters, kernel_size, activation='relu', input_shape=input_shape),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid'), # Default values
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(classes) # You can apply softmax activation here, see below for comentary
])

# As you can see, we can include a pooling layer with the simple line 'tf.keras.layers.MaxPooling2D'
# Pooling layers are always included after a convolutional layer.

# Important parameters of Pooling layers:
# - pool_size: Integer or tuple of 2 integers, window size over which to take the maximum.
#              (2, 2) will take the max value over a 2x2 pooling window. If only one integer is specified,
#              the same window length will be used for both dimensions. The most popular size is by far (2,2).
#
# - strides: Integer, tuple of 2 integers, or None. Strides values. Specifies how far the pooling window moves for each
#            pooling step. If None, it will default to pool_size.
#
# - padding: One of "valid" or "same" (case-insensitive). "valid" means no padding.
#            "same" results in padding evenly to the left/right or up/down of the input such that output has the same
#            height/width dimension as the input. Usually, no padding is necessary.
#
# For most problems, the default values are the ones we would like to use,
# so often we just write tf.keras.layers.MaxPooling2D()

# Finally, the 'classes' parameter specifies how many classes we have for the classification.
```

Compiling the model

```
# Defining the loss function

# In general, our model needs to output probabilities of each class,
# which can be achieved with a softmax activation in the last dense layer

# However, when using the softmax activation, the loss can rarely be unstable

# Thus, instead of incorporating the softmax into the model itself,
# we use a loss calculation that automatically corrects for the missing softmax

# That is the reason for 'from_logits=True'
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

# Compiling the model with Adam optimizer and the cathegorical crossentropy as a loss function
model.compile(optimizer='adam', loss=loss_fn, metrics=['accuracy'])
```

Defining early stopping callback

```
# Defining early stopping to prevent overfitting
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor = 'val_loss',
    mode = 'auto',
    min_delta = 0,
    patience = 2,
    verbose = 0,
    restore_best_weights = True
)
```

Training the model

```
# Train the network
model.fit(
    train_data,
    epochs = NUM_EPOCHS,
    callbacks = [early_stopping],
    validation_data = validation_data,
    verbose = 2
)

# Here, you need to provide train data and validation data, as well as specify for how many epochs to train.
```

Start your 365 Journey!