365 DataScience A very simple CNN network - Convolutional layer

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

# The outlined code below is to show how we can add a convolutional lay
er to a 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 followin
g 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', inp
ut_shape=input_shape),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(classes) # You can apply softmax activation h
ere, see below for comentary
])

# As you can see, we can include a convolutional layer with the simple
line 'tf.keras.layers.Conv2D'

# Important parameters of Convolutional layers:
# - filters: Integer, signifies how many filters/kernels to be included
 in the layer, thus, it controlls the output space.
#            Popular values - 32, 64, 128, 256, 512, 1024
#
# - kernel_szie: An integer or tuple/list of 2 integers, specifying the
 height and width of the 2D convolution window.
#                Can be a single integer to specify the same value for
all spatial dimensions.
#                Popular values - 3, 5, 7, 11
#
# - input_shape: Only specified in the first layer of the network. Indi
cates the shape ofthe input data.
#                Tensor with format '(batch_size, rows, cols, channels)
'. You can ommit the batch_size.
```

```
#                For example, the input shape for the MNIST dataset wou
ld be (28,28,1)

# Finally, the 'classes' parameter specifies how many classes we have f
or 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 lay
er

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

# 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=Tru
e)

# Compiling the model with Adam optimizer and the cathegorical crossent
ropy 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!