Keras results

The goal of this workflow is build a deep learning model to classify words. We will use the Speech Commands dataset which consists of 65,000 one-second audio files of people saying 30 different words. Here fit multiple Keras models to the dataset with different tuning parameters, pick the one with the highest classification test accuracy, and produce a trained model for the best set of tuning parameters we find. This eample is based on this and this blog post from the RStudio AI Blog and the targets-keras by W. Landau.

Many deep learning models are end-to-end, i.e. we let the model learn useful representations directly from the raw data. However, audio data grows very fast - 16,000 samples per second with a very rich structure at many time-scales. In order to avoid having to deal with raw wave sound data, researchers usually use some kind of feature engineering.

Every sound wave can be represented by its spectrum, and digitally it can be computed using the Fast Fourier Transform (FFT).

A common way to represent audio data is to break it into small chunks, which usually overlap. For each chunk we use the FFT to calculate the magnitude of the frequency spectrum. The spectra are then combined, side by side, to form what we call a spectrogram.

It's also common for speech recognition systems to further transform the spectrum and compute the Mel-Frequency Cepstral Coefficients. This transformation takes into account that the human ear can't discern the difference between two closely spaced frequencies and smartly creates bins on the frequency axis.

After this procedure, we have an image for each audio sample and we can use convolutional neural networks, the standard architecture type in image recognition models.

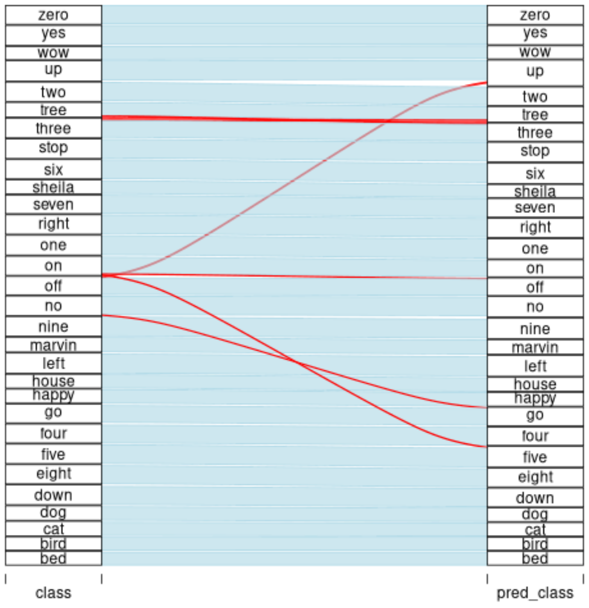
After data preprocessing (for more detailed specification see here) we definie the following model and train it:

```r
library(targets)
tar_read(model)
```

```
## Model
## Model: "sequential"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                     Param #
## ================================================================================
##  conv2d_3 (Conv2D)                  (None, 96, 255, 32)              320
##
##  max_pooling2d_3 (MaxPooling2D)     (None, 48, 127, 32)              0
##
##  conv2d_2 (Conv2D)                  (None, 46, 125, 64)              18496
##
##  max_pooling2d_2 (MaxPooling2D)     (None, 23, 62, 64)               0
##
##  conv2d_1 (Conv2D)                  (None, 21, 60, 128)              73856
##
##  max_pooling2d_1 (MaxPooling2D)     (None, 10, 30, 128)              0
##
##  conv2d (Conv2D)                    (None, 8, 28, 256)               295168
##
##  max_pooling2d (MaxPooling2D)       (None, 4, 14, 256)               0
```

```
##
##  dropout_1 (Dropout)              (None, 4, 14, 256)            0
##
##  flatten (Flatten)                (None, 14336)                 0
##
##  dense_1 (Dense)                  (None, 128)                   1835136
##
##  dropout (Dropout)                (None, 128)                   0
##
##  dense (Dense)                    (None, 30)                    3870
##
## ================================================================================
## Total params: 2,226,846
## Trainable params: 2,226,846
## Non-trainable params: 0
## ________________________________________________________________________________
```

A nice visualization of the resulting confusion matrix is to create an alluvial diagram:



We can see from the diagram that the most relevant mistake our model makes is to classify "tree" as "three". There are other common errors like classifying "go" as "no", "up" as "off". At 93% accuracy for 30 classes, and considering the errors we can say that this model is pretty reasonable.

The saved model occupies 25Mb of disk space, which is reasonable for a desktop but may not be on small devices. We could train a smaller model, with fewer layers, and see how much the performance decreases.