Capstone Project 2: Milestone Report II

As per Milestone Report I, we have extracted the data from the audio files and stored it as JSON file. This Json file has the following information about the audio input.

* Mappings
* Labels
* MFCCs
* Files

**Important Terms:**

1. MFCCs - Mel Frequency Cepstral Coefficients

They are coefficients that collectively make up an mel-frequency cepstrum (MFC). They are derived from a type of [cepstral](https://en.wikipedia.org/wiki/Cepstrum) (The cepstrum is the result of a mathematical transformation in the field of Fourier Analysis) representation of the audio clip (a nonlinear "spectrum-of-a-spectrum"). The difference between the [cepstrum](https://en.wikipedia.org/wiki/Cepstrum" \o "Cepstrum) and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal cepstrum. This frequency warping can allow for better representation of sound.

MFCCs are commonly derived as follows:

* Take the [Fourier transform](https://en.wikipedia.org/wiki/Fourier_transform) of (a windowed excerpt of) a signal.
* Map the powers of the spectrum obtained above onto the [mel scale](https://en.wikipedia.org/wiki/Mel_scale" \o "Mel scale), using [triangular overlapping windows](https://en.wikipedia.org/wiki/Window_function#Triangular_window).
* Take the [logs](https://en.wikipedia.org/wiki/Logarithm) of the powers at each of the mel frequencies.
* Take the [discrete cosine transform](https://en.wikipedia.org/wiki/Discrete_cosine_transform) of the list of mel log powers, as if it were a signal.
* The MFCCs are the amplitudes of the resulting spectrum.

Reference:

<https://en.wikipedia.org/wiki/Mel-frequency_cepstrum>

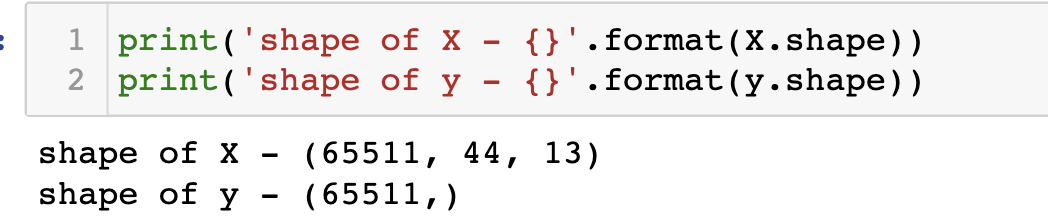
<https://medium.com/prathena/the-dummys-guide-to-mfcc-aceab2450fd>

1. n\_fft - nonuniform fast Fourier transform
2. hop\_length - how much we can advance the analysis time origin from frame to frame

Note: 22050 is the number of samples/ frames that a one second audio file can have

From the input json file, we are going to give MFCCs as input to deep learning model and train the model to predict the label of the audio file.

**Shape of inputs to the model:**



### Model Summary

**Training the model (40 epochs)**

### 

**Accuracy/ Loss Evaluation & Loss/ Epoch Graph:**

### 

**Recalling Deep Learning Concepts:**

1. Sequential Layer:

Sequential groups a linear stack of layers into a [tf.keras.Model](https://www.tensorflow.org/api_docs/python/tf/keras/Model). Sequential is

used to initialize the neural network.

1. 2D convolution layer:

This layer creates a convolution kernel that is convolved with the layer input to

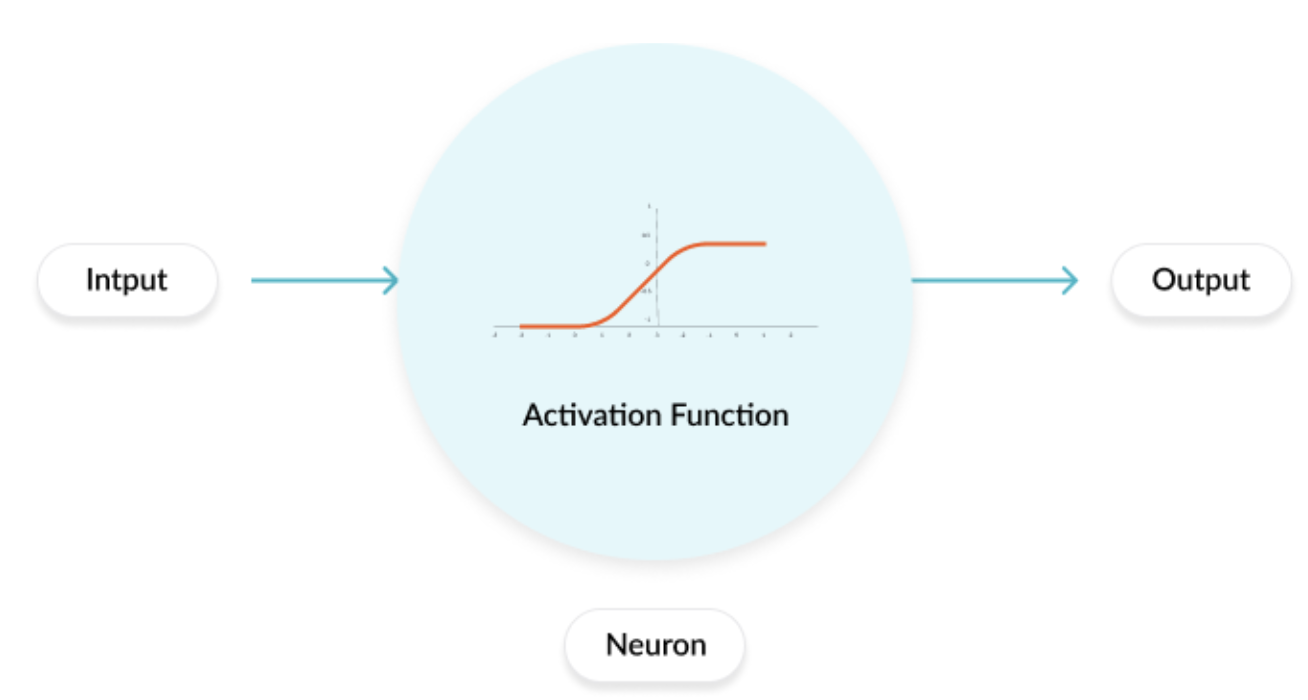
produce a tensor of outputs. If use\_bias is True, a bias vector is created and added to the outputs. Finally, if activation is not None, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide the keyword

argument input\_shape (tuple of integers, does not include the sample axis), e.g. input\_shape=(128, 128, 3) for 128x128 RGB pictures in data\_format="channels\_last".

1. Activation Function:

The function is attached to each neuron in the network, and determines whether it should be activated (“fired”) or not, based on whether each neuron’s input is relevant for the model’s prediction. Activation functions also help normalize the output of each neuron to a range between 1 and 0 or between -1 and 1.



Reference: <https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/>

Types of Activation Function:

1. Binary Step Function
2. Linear Activation Function
3. Non-Linear Activation Functions
   1. Sigmoid / Logistic
   2. TanH / Hyperbolic Tangent
   3. ReLU (Rectified Linear Unit)
   4. Leaky ReLU
   5. Parametric ReLU
   6. Softmax
   7. Swish
4. Kernel\_Regularizer:

Regularizer function applied to the kernel weights matrix

More on Regularization (l1 & l2 regularization):

<https://towardsdatascience.com/regularization-techniques-and-their-implementation-in-tensorflow-keras-c06e7551e709>

1. BatchNormalization:

Training is impeded by vanishing gradients, which occurs when a network stops updating because the gradients, particularly in earlier layers, have approached zero values. Incorporating Xavier weight-initialization and ReLu activation functions helps counter the vanishing gradient problem. These techniques also help with the opposite, yet closely related issue of exploding gradients, where the gradients become extremely large preventing the model from updating.

Perhaps the most powerful tool for combatting the vanishing and exploding gradients issue is Batch Normalization. Batch Normalization works like this: for each unit in a given layer, first compute the z score, and then apply a linear transformation using two trained variables 𝛾 and 𝛽. Batch Normalization is typically done prior to the non-linear activation function (see below figure), however applying it after the activation function can also be beneficial.

Reference:

<https://towardsdatascience.com/how-to-use-batch-normalization-with-tensorflow-and-tf-keras-to-train-deep-neural-networks-faster-60ba4d054b73>

1. MaxPooling2D:

When added to a model, max pooling reduces the dimensionality of images by reducing the number of pixels in the output from the previous convolutional layer.

Reference:

<https://youtu.be/ZjM_XQa5s6s>

1. Flatten:

Flatten is the function that converts the pooled feature map to a single column

which is then fed to the neural network for processing.

1. Dense Layer:

Dense adds the fully connected layer to the neural network.

1. Optimizer:

Optimizers tie together the loss function and model parameters by updating the model in response to the output of the loss function. In simpler terms, optimizers shape and mold your model into its most accurate possible form by futzing with the weights. The loss function is the guide to the terrain, telling the optimizer when it’s moving in the right or wrong direction.

Available optimizers

* + [SGD](https://keras.io/api/optimizers/sgd)
  + [RMSprop](https://keras.io/api/optimizers/rmsprop)
  + [Adam](https://keras.io/api/optimizers/adam)
  + [Adadelta](https://keras.io/api/optimizers/adadelta)
  + [Adagrad](https://keras.io/api/optimizers/adagrad)
  + [Adamax](https://keras.io/api/optimizers/adamax)
  + [Nadam](https://keras.io/api/optimizers/Nadam)
  + [Ftrl](https://keras.io/api/optimizers/ftrl)

Reference: <https://algorithmia.com/blog/introduction-to-optimizers>

<https://towardsdatascience.com/a-quick-guide-to-neural-network-optimizers-with-applications-in-keras-e4635dd1cca4>

1. Loss Function:

As part of the optimization algorithm, the error for the current state of the model must be estimated repeatedly. This requires the choice of an error function, conventionally called a loss function, that can be used to estimate the loss of the model so that the weights can be updated to reduce the loss on the next evaluation.

Reference:

<https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/>