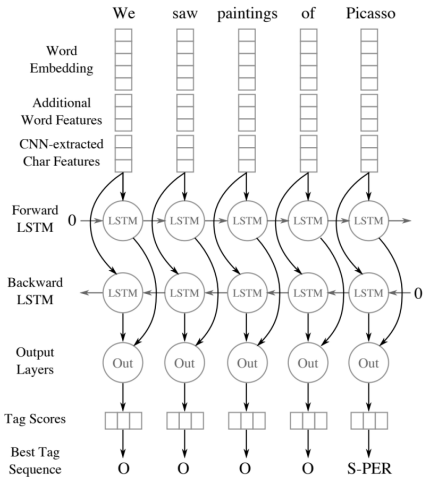
Natural Langauge Processing

Spark NLP:

- Open-source NLP library built on top of Apache Spark and Spark ML.
    - Apache Spark:
        - Unified analytics engine for large-scale data processing.
    - Spark ML
        - Provide a uniform set of high-level APIs that help users create and tune practical machine learning pipelines
- Comes with 1100+ pretrained pipelines and models in more than 192 languages.
- It supports nearly all the NLP tasks and modules that be used seamlessly in a clusters
- Model Architecture
    - Named Entity Recognition
    - 

        - Uses Bi-LSTM+Char-CNN+CRF+Word Embeddings
            - 50 Word Embeddings include BERT, Small BERT, BioBERT, CovidBERT,.etc
                - BERT:
                    - Bidirectional Encoder Representations-> transformer-based ml learning technique for NLP
    - Multi-class text classification
        - Detect emotions, cyberbullying, fake news, etc.
    - Multi-label text classifications
        - Detect toxic comments, movie genre, etc.

    - 90 + pretrained Word and Sentence Embedding models
    - Universal Sentence Encoder as an input for text classification

- Developed to be a single unified solution for all the NLP tasks and the only Library that can scale up for training and inference in any Spark cluster.
- Written in Scala and provides open-source APIs in Python, Java, Scala, and R, easy API to integrate with ML pipelines, commercially supported by John Snow Labs.
- Spark NLP's annotators utilize rule-based algorithms, machine learning, and deep learning models which are implemented using Tensorflow (optimized for accuracy, speed, scalability, and memory utilization).
- Annotators
    - Estimators
        - Fit on a data frame to produce a transformer
            - ie) Learning algorithm which trains on a data frame and produces a model
    - Transformers
        - Transform one data frame into another data frame
            - ie)ML model that transforms a data frame with features into a data frame with prediction
    - AnnotatorApproach
        - Extends estimators from Spark ML, trained through fit
    - AnnotatorModel
        - Extends transformers which are meant to transform data frames through transformers.
- Can be used offline and deployed in air-gapped networks
    - No need to worry about exposing the sensitive information(Protected Health Information)
- Two versions
    - Open-source
        - All the features and components expected from any NLP
    - Enterprise
        - Licensed and designed towards solving real-world problems in the health care domain and extends the open-source version
        - Modules to help researchers and data practitioners: Named entity recognition(NER), assertion status(negativity scope) detection, relation extraction, entity resolution

(SNOMED,RxNorm, ICD10,etc), clinical spell checking, contextual parser, text2SQL, deidentification and obfuscation.

- Named Entity Recognition in Clinical NLP pipeline
    - Primary building block in text mining systems.
        - In medical domain
            - Recognizes the first meaningful chunks out of a clinical note-> fed down as an input to a subsequent downstream task -> ie): clinical assertion status detection, clinical entity resolution and de-identification of sensitive data -> assign an assertion status to each named entity given its context, status of an assertion explains how a named entity such as clinical finding, procedure, relate to the patient by assigning a label such as present("patient is diabetic"), absent("patient denies nausea"), conditional ("dyspnea while climbing stair") or something else.
- Specialized in processing biomedical and clinical text(Using NER)
    - Electronic Health Records: the primary source of information for clinicians tracking the care of their patients
        - Most of the time information in these records is unstructured and largely inaccessible for statistical analysis. However, with the help of NER, we can now extract relevant information from unstructured EHR.
- Production-ready models/tools in the biomedical and clinical domain in NLP research and NER systems.
- Clinical NLP researchers can implement the latest algorithms into their workflows and start using them immediately.

- Flow Diagram of a Spark NLP Pipeline

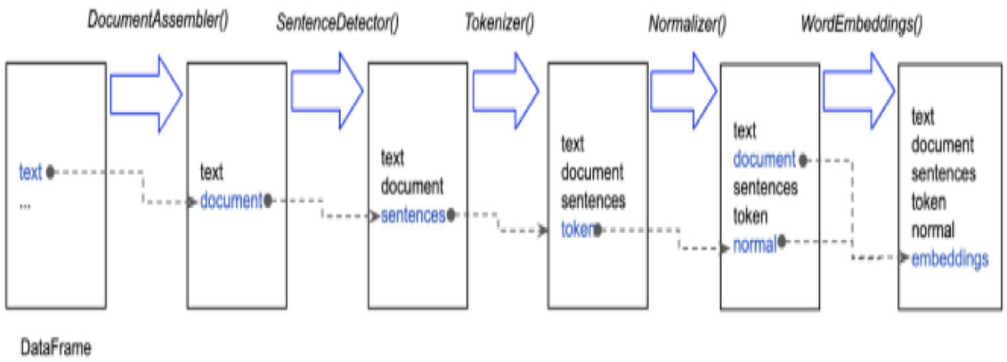
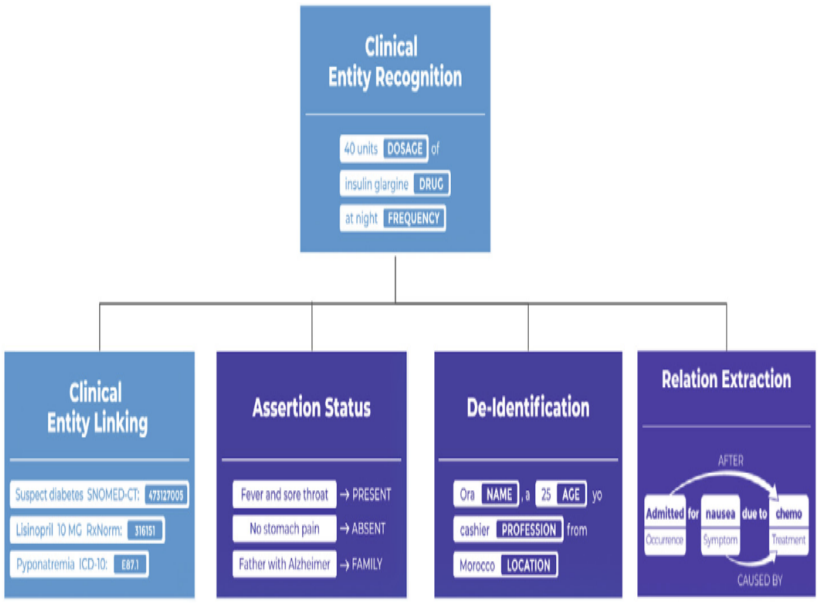


Fig. 1. The flow diagram of a Spark NLP pipeline. When we fit() on the pipeline with a Spark data frame, its text column is fed into the DocumentAssembler() transformer and a new column *document* is created as an initial entry point to Spark NLP for any Spark data frame. Then, its document column is fed into the SentenceDetector() module to split

- Named Entity Recognition
    - The building block of medical text mining pipelines

assertion model



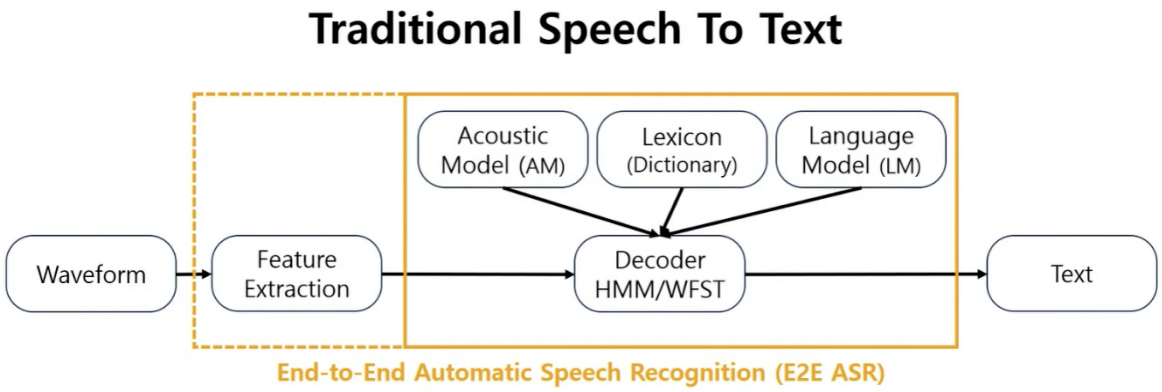
Speech recognizer

- Imagage Classification : 1->1
- Speech recognition: N->1
    - Sound recording has various variables (environmental, how it is collected, etc..),
- When generating Waveform
    - Variety
        - Styles: Reading, conversation, spontaneous,
        - Variance: Disfluency, Stuttering, mic quality, channels
        - Noise: Outdoor, room, school, subway
- Collecting , Inspecting, managing, data-> hard
- Waveform -> Spectrogram -> Text
    - Data Loss
        - Waveform -> Spectrogram
            - Phase

        - Spectrogram -> Text
            - Noise, variance, style

Traditional Speech recognition

- Extracted Feature with Fourier Transform/Mel-Frequency Cepstral Coefficients
- Based on Hidden Markov Model
    - Three different objective function has to train independently
        - Pronunciation Model: Mapping word and Phoneme sequence. (Lexicon)
        - Acoustic Model: predicting phoneme and voice's relation. (Hidden Markov + Gaussian Mixture Model)
        - Language Model: Probability of word sequence occurring. (n-gram model)
- Acoustic Model = Hidden Markov Model + Gaussian Mixture Model
- Train Acoustic and Language models independently



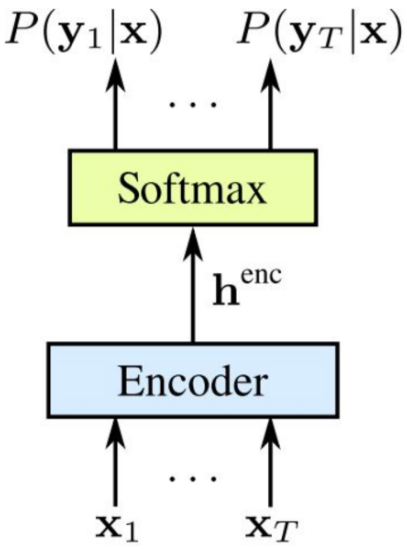
- End-to-End Deep Learning
    - Rule-Based + probabilistic Model
    - Pre-Process/Post-Process is directly related to performance

- Problems with Speech-To-Text
    - Hard to Label -> Difficult to do supervised-learning
        - Obscure boundaries
        - Hard to provide an answer for each Frame

- Three different ways for End-to-End System
    - Utilizing CTC (Deep Speech 2)

- Utilizing Attention Network (Listen,Attend,and Spell(LAS))
- Utilizing RNN-Transducer (RNN + CTC Loss)

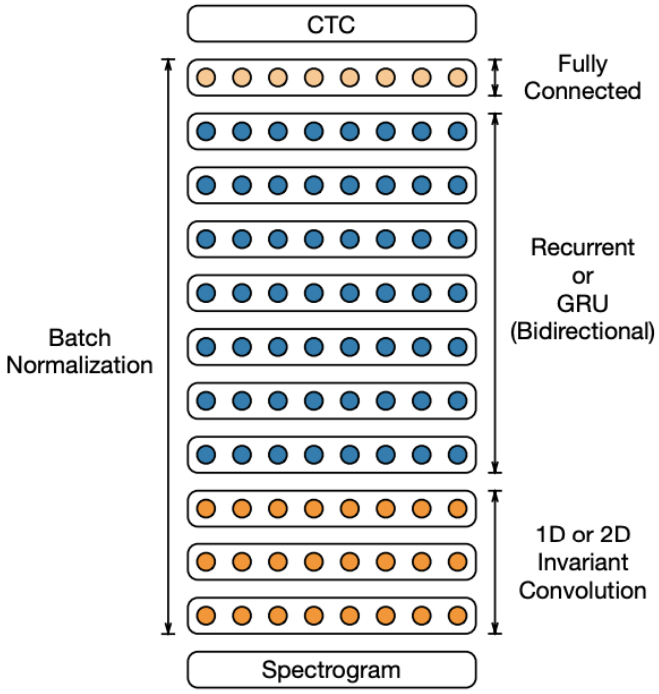
Neural Acoustic Model (CTC: Connectionist Temporal Classification)



- Allows performing speech recognition without target word/phoneme sequence alignment.
- Placed at the end of model-> loss and gradient layer.
- Expand the Text length (T) according to the Sound length(S).
- Pro
  - Could model with an encoder only, no need for a decoder
    - Encoding could be done using CNN, LSTM/GRU, Transformer block

- ie) S=7, T= 3(“CAT”)
  P(T|S) = P(_C_A_T)+ P(__CA__T)+
  P(_CC_A_T_) +...

Deep Speech 2



-
    - Input Power Spectrogram -> Extract features with CNN -> Bi-driectional RNN -> FC-Layer.
        - Batch Normalization was used in between layers
- Structure
    - Time-Domain Convolution

$$h_{t,i}^{l} = f\left(w_i^l \circ h_{t-c:t+c}^{l-1}\right)$$

        -
    - Clipped ReLu for the activation

$$f(x) = \min\{\max(x, 0), 20\}$$

        -
    - Bi-direction RNN (Using Vanilla RNN or LSTM)

$$\overrightarrow{h_t^l} = g\left(h_t^{l-1}, \overrightarrow{h_{t-1}^l}\right)$$
$$\overleftarrow{h_t^l} = g\left(h_t^{l-1}, \overleftarrow{h_{t+1}^l}\right)$$

        -

$$\overrightarrow{h_t^l} = f\left(W^l h_t^{l-1} + \overrightarrow{U^l h_{t-1}^l} + b^l\right)$$

- Fully Connected Layer

$$h_t^l = f\left(W^l h_t^{l-1} + b^l\right)$$

- Softmax Layer

$$p\left(l_t = k|x\right) = \frac{\exp\left(w_j^L h_t^{L-1}\right)}{\sum_j \exp\left(w_j^L h_t^{L-1}\right)}$$

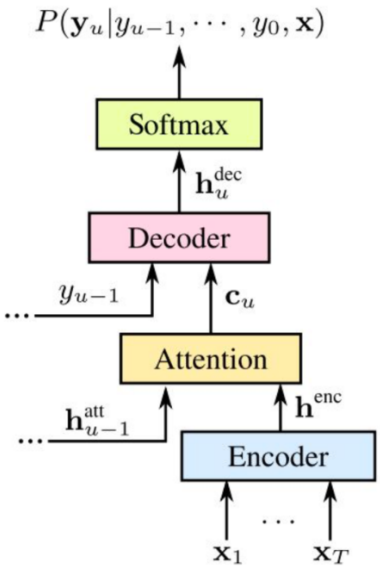
  - Calculates CTC loss with an output of the above function and true value

- Batch Normalization for RNN

$$\overrightarrow{h_t^l} = f\left(\mathcal{B}\left(\mathcal{W}^l h_t^{l-1}\right) + \overrightarrow{\mathcal{U}^l h_{t-1}^l} + b^l\right)$$

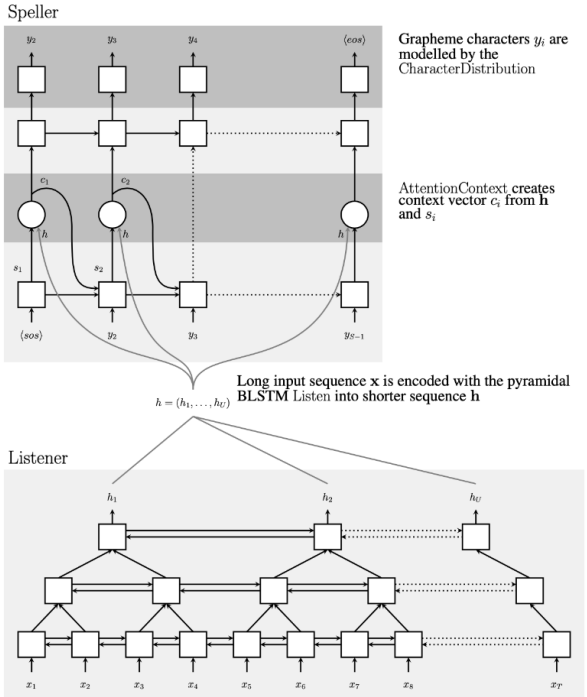
- SortaGrad
  - Adjusted the minibatch to fit CTC loss.
  - CTC loss: Increase in speech frame sequence, more loss, and increase in gradient
- For alignment purposes, using a bi-directional RNN layer using CTC loss.

Neural Acoustic Model (Attention-based Encode-Decoder Models)



- Consisted of encoder, attention, and decoder(Similar to Acoustic, Alignment, Language model)
    - Encoder: Sound -> Hidden Vector
    - Attention: allow the decoder to target which specific information from the encoder
    - Decoder: generate target sequence

Listen, Attend, and Spell



- Architecture

- Based on Sequence to Sequence Model
- Encoder: Listener, Decoder: Speller
- Listener: Pyramid RNN
  - Generally, there are a lot of voice frames, therefore, halve the frame in between layers to obtain meaningful information.
  - Takes an input of log-mel filter banks sequence.
  - Uses bi-directional pyramid LSTM.
    - Halve the input voice sequence per layer

$$h_i^j = \text{pBLSTM}\left(h_{i-1}^j, \left[h_{2i}^{j-1}, h_{2i+1}^{j-1}\right]\right)$$

- Attention
  - Connecting between listener and speller
  - Informs speller to focus on which part of input when generating text sequence.

$$e_{i,u} = \langle \phi(s_i), \psi(h_u) \rangle$$

$$\alpha_{i,u} = \frac{\exp(e_{i,u})}{\sum_u \exp(e_{i,u})}$$

$$c_i = \sum_u \alpha_{i,u} h_u$$

- Attend and spell

$$s_i = \text{RNN}\left(s_{i-1}, y_{i-1}, c_{i-1}\right)$$

$$c_i = \text{AttentionContext}\left(s_i, \mathbf{h}\right)$$

$$P\left(y_i | \mathbf{x}, y_{<i}\right) = \text{CharacterDistribution}\left(s_i, c_i\right)$$

  - Speller
    - 2 forward directional LSTM layer
- Learning
  - Training is performed by the model's output(character distribution) getting close to the answer's distribution. Uses Cross-Entropy for a loss function.

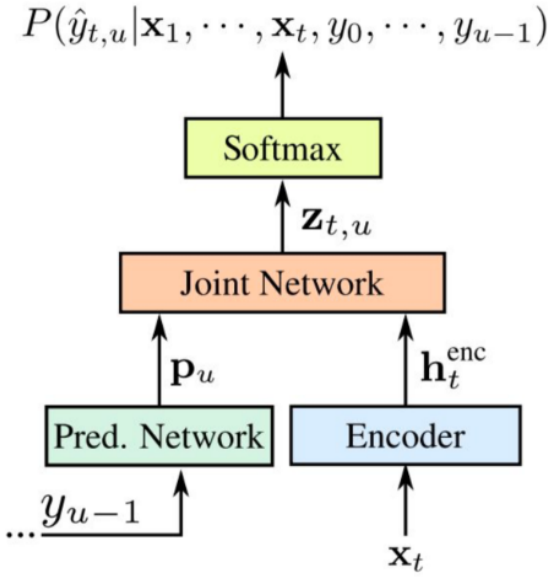
- Decoding
  - Done with beam search.
    - For each step pick β pamount of sequence candidate-> <eos> -> delete sequence from beam and put it to final decoded lists. Continue beam search until reaching pre-determined amount

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \log P(\mathbf{y}|\mathbf{x})$$

  -
  - Rescoring

$$s(\mathbf{y}|\mathbf{x}) = \frac{\log P(\mathbf{y}|\mathbf{x})}{|\mathbf{y}|_c} + \lambda \log P_{\text{LM}}(\mathbf{y})$$

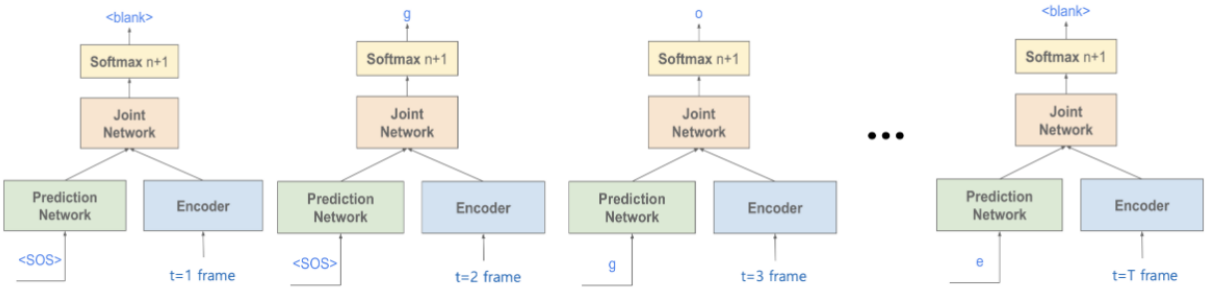
Neural Acoustic Model (Recurrent Neural Network Transducer)



-
- RNN based language model + CTC loss
- RNNT
  - Online model
    - Can predict during the voice input.
- Consisted of Encoder, Prediction network, and Joint network
- Encoder

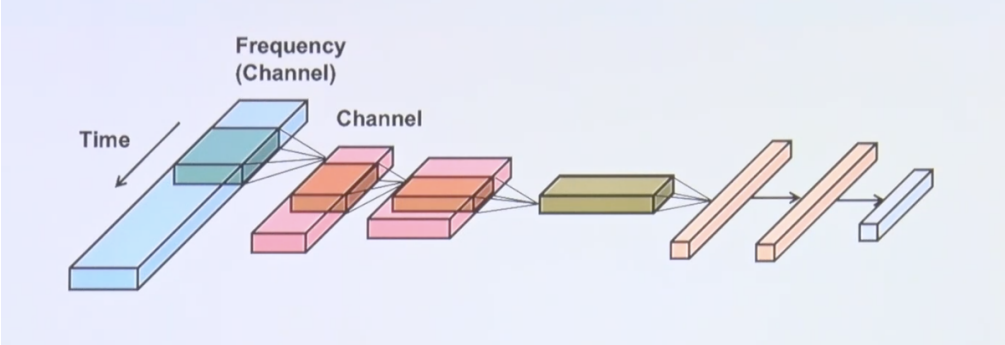
    - From beginning to an input sequence's point present, t's voice pitcher sequence -> encode to hidden vector h^{enc}_{t}
- Prediction
    - From beginning to output sequence's point present, u-1's output word/phoneme sequence information is converted into vector p_u
- Joint
    - Input encoder, and prediction network's results. Output u_{th} word/phoneme logit vector.

- Inference:



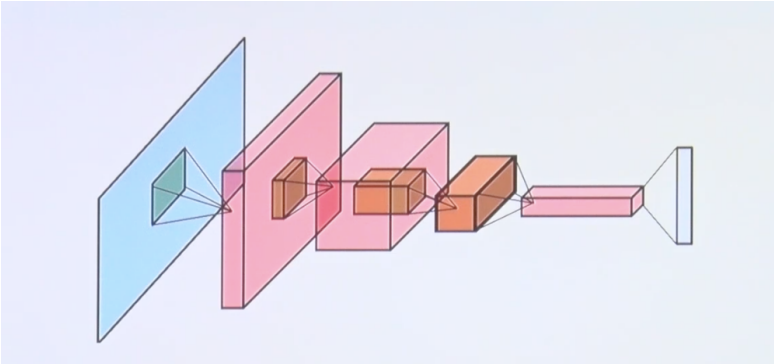
CNN in audio

1-D CNN



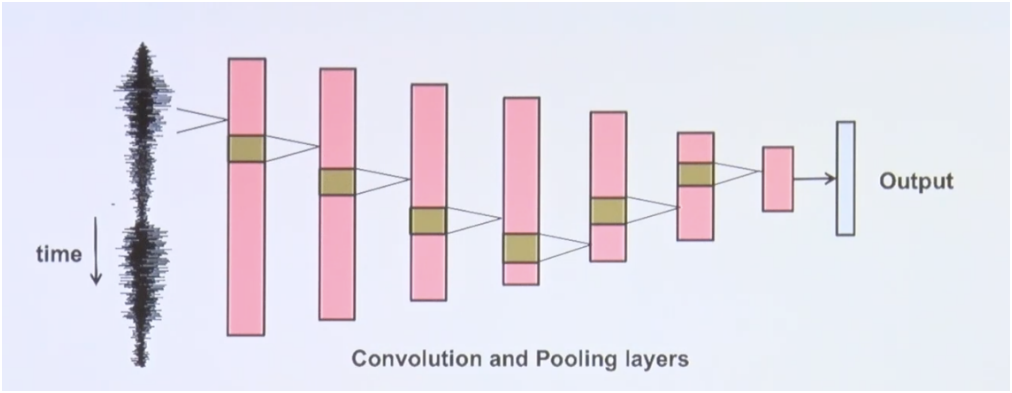
- Audio -> X channel
- 1Convolution Filter size: Fixed Frequency, Time-level 1-D Convolution
- Extract multiple features by adjusting the time. Time-specific features

2-D CNN



- Convolution filter size: Frequency, Time
- Using Frequency and Time to stack convolutional layer.
- As layer increments-> Output channel
- Find patterns in Time and Frequency

Sample CNN

- 

- Characteristic

    - Allows to use of input data as wave-form
    - Phase invariant representation is not loss (which happens during conversion to spectrogram)
    - The kernel calculates the spectral bandwidth in regards to input

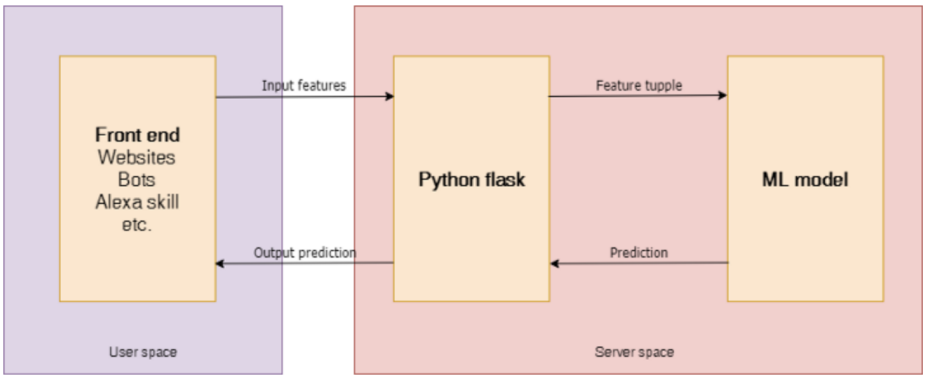
Vosk

- Offline open source speech recognition toolkit.
- Speech recognition for 20+ languages
- Support various programming languages like Python, Java, Node.JS, and others
- Speech recognition for chatbots, smart home appliances, virtual assistants
- Can also create subtitles for movies and transcription for lectures and interviews.
- Support small devices like Rasberry PI, smartphones to Big Clusters.
- Information sources in speech recognition
    - The traditional data source that Vosk used to build the recognition graph
        - Acoustic Model - Model of sounds of the language
        - Language Model - Model of word Sequences
        - Phonetic dictionary - Model of the decomposition of words to sounds
            - Pros:
                - Quickly replace the knowledge source(introduce new words with non-standard pronunciation such as technical terms)
                - Train model on one domain and use for another domain just by replacing language model
                - Tune acoustic model separately
                - The system can be compact and able to learn from pure texts
            - Con:
                - Words are compiled statically and it is not easy to introduce new words
    - Newer systems use a Neural network that compiles both language model, pronunciation, and acoustic model into a single monolithic model.
        - DNN-HMM architecture.

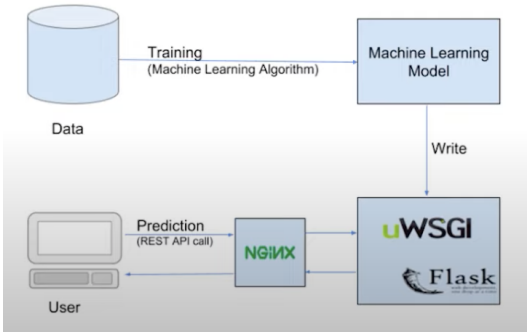
        - DNN is used for sound scoring(acoustic scoring), HMM and WFST are used for language models

Deployment: Flask API

- Flask
    - Web Server Gateway Interface (WSGI)
        - Used to forward requests from a web server to a backend Python web application or framework.
    - Simple/Light version of web framework
    - Allows to interact with python code
    - Like Java's Spring, the Python version of the web framework.
    - Used for web application library
    - It can be used as a bridge where it can connect the machine learning model and front-end web page.



ML-Model Deployment using Flask

- Structure
  - 

  -
  -
  - ML model.py
  - App.py
    - Flask API: Receives input through API calls, computes and predicts values with our model then returns
  - HTML/CSS
    - Front-End, HTML template and CSS style for a user to interact

ie)App.py:

- Local ML deployment using Flask
  - Basic Structure
  -

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World!'

if __name__ == '__main__':
    app.run()
```

ex) : App.py

```
from flask import Flask , render_template ,request
import os
import matplotlib.pyplot as plt
import style_transfer
```

- Import necessary library for a deployment
  - style_transfer: ML model which we want to deploy

@ app. route('/') = extension of an URL, which URL will perform an execution.

- By leaving it as ("/") it will automatically bring the user to index.html

```
@app.route("/")
def home():
    return render_template("index.html")
```

- The routing decorator connects the function with the URL.
- If we want to get the HTML template, need to import render_template.
- Index.html files, in this case, the form type was an image, so used multipart/form-data, where none was encoded. (vary from input datatype)

```
<form action="/success" method="post" enctype="multipart/form-data" >
```

- On Flask runnable app.py, use Post methods to obtain input from a user and print the results. Once the file is uploaded, @ app. route("/success"~) is executed.

```
@app.route("/success", methods=['POST'])

def upload_file():
```

- In this case, another HTML page was used to show the outputs of the results of ML inference. It was titled success.html.

Index.html

```
    <span class="btn btn-default btn-file">
      <input type="file" name="file" required>
    </span>

  <input type="submit" value="Upload your image" class="btn btn-primary" required>
</form>
</div>
```

- Allow users to submit the file.

App.py

- Then, write a code that processes the input data obtained from the users according to an ML model, in this case, an image.
- Using the requests. files to bring the file and save the file at the designated path on the server (app. config was used)

```
content = request.files['file']
style = request.form.get('style')
content.save(os.path.join(app.config['UPLOAD_FOLDER'], 'content.jpg'))
#load in content and style image
content = load_image('./static/image/upload'+content.filename)
#Resize style to match content, makes code easier
style = load_image('./static/image/'+ style+'.jpg', shape=content.shape[-2:])
vgg = model()
target = stylize(content,style,vgg)
x = im_convert(target)
plt.imsave(app.config['UPLOAD_FOLDER']+'target.png',x)

return render_template('success.html')
```

- Once done, by returning with render_template, it brings the users to Success.html.

Success.HTML

- ```

    <h3>content image</h3>
    <img src="../static/image/upload/content.jpg" class="img-responsive" style="width:10

    <h3>styled image</h3>
    <img src="../static/image/upload/target.png" class="img-responsive" style="width:100
  ```
- To run, we simply need to run the Flask app.py, then it would run on a local port.
  ```