دانشگاه تهران

پردیس فارابی

دانشکده مهندسی

گروه مهندسی کامپیوتر

پیاده سازی سیستمی برای تشخیص کلمات کلیدی فارسی در صوت با استفاده از متد های یادگیری عمیق

نگارش:

محمدرضا افشاری

استاد راهنما:

دکتر زهرا موحدی

گزارش پروژه برای دریافت درجه ی کارشناسی در رشته مهندسی کامپیوتر

شهریور ۱۴۰۱


بسم الله الرحمن الرحيم

چکیده

امروزه با پیشرفت تکنولوژی و فناوری، زندگی ما به سمت دیجیتالی شدن پیش میرود. دستیار صوتی هوشمند یکی از فناوری های نسبتا جدید است که کار شما را برای کنترل ابزارهای مختلف مانند تلفن همراه هوشمند، تبلت، لپ تاپ و حتی تلویزیون و برخی دیگر از دستگاه های الکترونیکی آسان تر میکند.

تنها کافی ست نیاز ها و خواسته های خود را بیان کنیم تا دستیار صوتی هوشمند به دستور های ما عمل کند. بنابراین این برنامه با گوش کردن صدای شما، هرآنچه را که بخواهید انجام میدهد.

در این پروژه، با استفاده از دیتاست های موجود در اینترنت و جمع آوری دیتا و استخراج ویژگی های مورد نیاز از سیگنال های صوتی و استفاده از مدل های یادگیری عمیق CNN و RNN، مدلی برای تشخیص کلمات کلیدی در صوت را ایجاد کردیم که قادر به تشخیص کلمات و مشخص کردن محل رخ داد آن در صوت میباشد.

در نهایت یک برنامه ی دسکتاپ برای نمایش گرافیکی صوت و رابط کاربری گرافیکی ساده پیاده سازی شد.

کلمات کلیدی: بازشناسی الگو، پردازش گفتار، یادگیری عمیق، سیگنال صوتی، cnn، rnn، برنامه دسکتاپ

فهرست

۶ فصل ششم

فصل اول

۱-۱ مقدمه

مهندسان و دانشمندان پدیده ارتباط گفتاری را با چشم اندازی به ایجاد سیستم های کارآمدتر و مؤثرتر ارتباط با انسان و انسان با ماشین مطالعه کرده اند. با شروع دهه ۱۹۶۰، پردازش سیگنال دیجیتال نقش اصلی را در مطالعات گفتار بر عهده گرفت و امروزه پردازش گفتار کلید تحقق ثمرات دانشی است که طی دهه ها تحقیق به دست آمده است. به زبان دیگر پردازش گفتار به توانایی یک ماشین یا برنامه برای شناسایی کلماتی که انسان ها صحبت میکنند گفته میشود.

سیستم تشخیص و پردازش گفتار از بخش های مختلفی تشکیل شده که این بخش ها شامل: دیجیتالی کردن صوت به صورتی که قابل پردازش برای کامپیوتر باشد، تقسیم بندی صوت، تجزیه و استخراج ویژگی ها از صوت، تحلیل صوت و تشخیص الگوها در آن با استفاده از الگوریتم مناسب اشاره کرد.

از کاربرد های پردازش صوت و گفتار میتوان به دستیار های صوتی، خانه های هوشمند، وسایل الکترونیکی که قادر به تشخیص چند دستور محدود هستند و ... اشاره کرد.

از انواع این سیستم ها به سیستم تشخیص کلمه کلیدی برای کاربرد های محدود (برای مثال تشخیص کلمه خاص برای روشن شدن دستگاه یا کنترل دستگاه با صوت)، دستیار های صوتی با کاربرد های مختلف، تبدیل صوت به متن، ترجمه ی همزمان گفتار اشاره کرد که این سیستم ها در زندگی امروزه نقش بسزایی را بازی میکنند.

با پیشرفت های صورت گرفته در حوزه ی هوش مصنوعی و یادگیری عمیق، این سیستم ها به عملکردی نزدیک و حتی بالاتر و بهینه تر از انسان ها دست یافته اند و با زیاد شدن دیتا و جمع آوری هدف دار آنها توسط شرکت ها، این پیشرفت ها روز به روز چشم گیر تر میشوند.

هدف این پروژه ایجاد سیستمی برای تشخیص کلمات کلیدی مشخصی در صوت میباشد که شامل گام های زیر برای انجام آن میشود:

۱. جمع آوری داده ها – جمع آوری داده های خام موجود در اینترنت و جمع آوری آنها به صورت دستی
۲. تبدیل داده های خام به داده های قابل استفاده – پاکسازی و استخراج بخش های مفید دیتاست ها و لیبل زدن آنها
۳. پیش پردازش داده ها و ایجاد دیتاست – تبدیل داده ها به فرمت قابل استفاده برای مدل های یادگیری عمیق و جمع آوری آنها در یک دیتاست واحد برای سهولت در فاز train مدل
۴. مشخص کردن معماری مدل ها – مشخص کردن جزئیات مدل ها

۵. پیاده سازی مدل ها و train آنها
۶. بهینه سازی عملکرد مدلها
۷. استفاده از چند نخی برای کار با مدل در کنار برنامه اصلی
۸. پیاده سازی برنامه دسکتاپ

در ادامه به مرور کارهای مشابه و شرح گام های بالا خواهیم پرداخت.

فصل دوم

۲-۱ مروری بر کارهای مشابه

با پیشرفت هوش مصنوعی و یادگیری عمیق و همچنین بالا رفتن توانایی پردازش سخت افزار ها و بهینه سازی الگوریتم های یادگیری ماشین، پیشرفت های زیادی در زمینه پردازش صوت صورت گرفته است. در ادامه مقالاتی که در این زمینه کار شده آورده شده است.

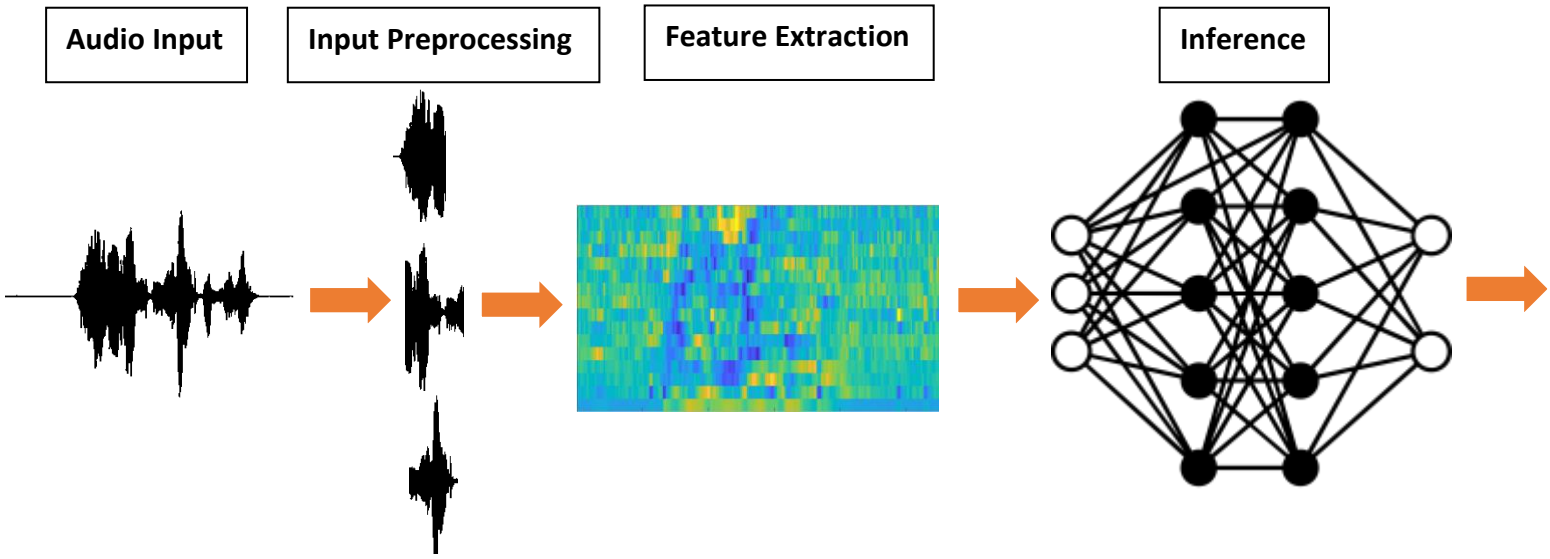
ردیف	شماره مرجع	محتویات پژوهش
۱	[1]	مروری بر روش ها و پیشرفت های یادگیری عمیق در زمینه ی پردازش صوت و بررسی تکنیک های به روز استفاده شده در دنیای واقعی
۲	[2]	ارائه روش بهینه برای سنتز صوت و تبدیل متن به صوت با استفاده از روش های یادگیری عمیق و مدل های شبکه عصبی بازگشتی
۳	[3]	ارائه مدلی برای تشخیص فعالیت صوتی و مکالمه در صوت و استخراج آن
۴	[4]	استفاده از یک سیستم برای تشخیص دو زبان متفاوت انگلیسی و چینی با استفاده از یادگیری عمیق
۵	[5]	تشخیص صدای گریه نوزاد توسط یادگیری عمیق

از دیگر موارد دیگر میتوان به دستیار های صوتی اشاره کرد. برای مثال دستیار صوتی شرکت آمازون که Alexa نام دارد و یا دستیار صوتی شرکت اپل که Siri نام دارد و یا دستیار صوتی گوگل که Google assistant نام دارد. این دستیارهای صوتی با توجه به دسترسی این شرکت ها به مقادیر حجیم دیتا عملکرد بسیار خوبی دارند و با داشتن یک مدل زبان انگلیسی، قابلیت پردازش بر روی گفتار را دارند و میتوانند دستورات کاربر را اجرا کنند و باعث سهولت استفاده تلفن های هوشمند میشوند. از دیگر موارد استفاده از پردازش صوت میتوان به دستیار های صوتی برای افراد نابینا اشاره کرد که با خواندن اطلاعات روی صفحه نمایش تلفن همراه یا سیستم های خانگی با لحنی نزدیک به انسان، امکان استفاده از این محصولات را به این افراد میدهند. از دیگر موارد استفاده سیستم های تشخیص wake word در لوازم الکترونیکی میباشد. برای مثال در اکثر کنسول های بازی امکان روشن شدن این کنسول ها با آوردن نام آنها و یا گفتن کلمه ای دیگر، وجود دارد.

فصل سوم

۳-۱ شرح کلی سیستم

در این قسمت به شرح کلی سیستم و توضیح الگوریتم های مورد استفاده میپردازیم. در سیستم پیاده سازی شده این پروژه، داده های ورودی از طریق میکروفون یا بصورت فایل های صوتی به سیستم داده میشود و سپس یک سری اقدامات بر روی این داده ی ورودی انجام میگیرد تا این داده ها قابل استفاده توسط مدل های ما شوند. در این بخش ابتدا سیگنال صوتی به بخش هایی با اندازه مساوی تقسیم میشود (در این پروژه به قسمت های ۱ ثانیه ای). این تقسیم بندی به صورتی انجام میشود که قسمت ها با هم اشتراک داشته باشند تا اگر کلمه ی مورد نظر مدل در این صوت گفته شده بود در این قسمت بندی از بین نرود و برای مثال نیمی از آن در یک قسمت و باقی آن در قسمت دیگر نیفتد. جلوتر در رابطه با روش دقیق این کار توضیحات تکمیلی خواهم آورد. بعد از این تقسیم بندی، ویژگی ها از این قسمت ها استخراج میشوند و آماده ی استفاده برای مدل ها میشوند. سپس این ویژگی های استخراج شده به عنوان ورودی به مدل داده میشود تا مدل پیشبینی را انجام دهد. پس از آن خروجی به کاربر به نمایش در آورده میشود.



System's Pipeline

در فصل های بعدی به تفکیک و با جزئیات به فعالیت های انجام شده برای این پروژه خواهیم پرداخت. در ادامه کد های این پروژه و فصل مورد نظر آنها را مشاهده میکنید:

- کد convert.py برای تبدیل فایل های با فرمت mp3 به wav. تا قابل پردازش توسط کتابخانه ها و پایتون باشند. فصل ۴
- کد label.py برای لیبل زدن فایل های صوتی. فصل ۴
- کد data_prep.py برای بررسی کلی داده ها. فصل ۴
- کد extract_keys.py برای جداسازی فایل هایی که حاوی کلمه کلیدی هستند. فصل ۴
- کد Record.py برای ضبط صدا و جمع کردن دستی دیتا. فصل ۴
- کد model.py برای پیاده سازی مدل ها. فصل ۵

- کد predict.py برای انجام پیشبینی ها توسط مدل ها. فصل ۵
- کد upsample.py برای رفع مشکل نامتوازن بودن دیتاست ها. فصل ۴
- کد make_dataset.py برای ایجاد دیتاست های قابل استفاده توسط مدل ها. فصل ۴

۳-۲ بررسی الگوریتم های یادگیری عمیق

در این بخش به بررسی الگوریتم های مختلف یادگیری عمیق و استفاده شده در این پروژه خواهیم پرداخت.

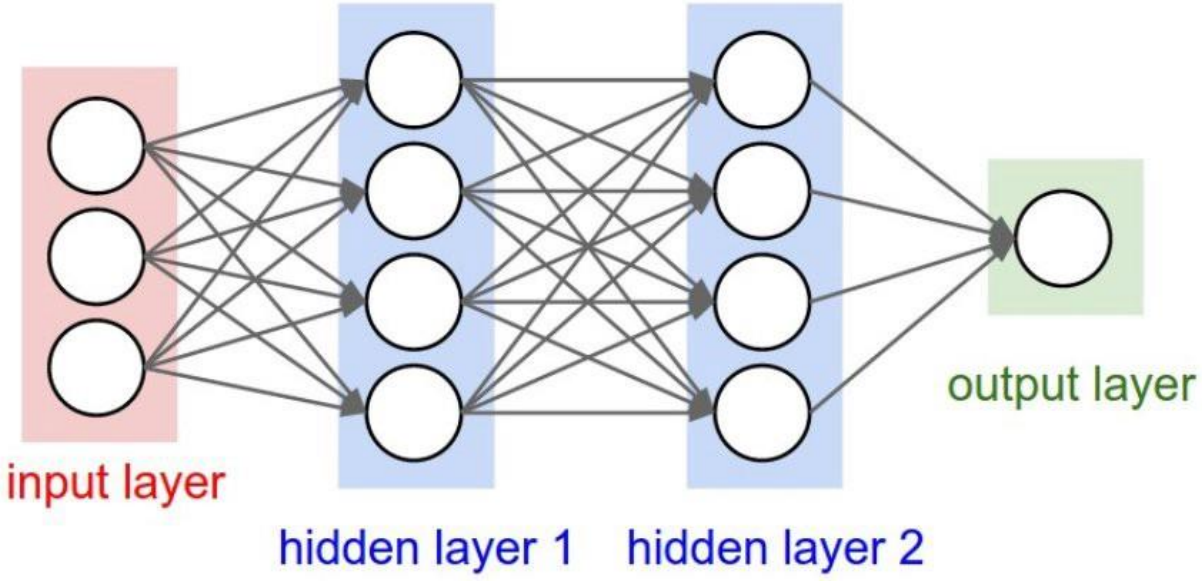
در این پروژه از الگوریتم های یادگیری عمیق برای ایجاد مدل ها استفاده شده. از جمله این مدل ها شبکه عصبی کانوولوشنال و شبکه عصبی بازگشتی هستند که در این پروژه مورد استفاده قرار گرفته اند.

یادگیری عمیق یک زیرشاخه از یادگیری ماشین و برمبنای مجموعه ای از الگوریتم ها است که در تلاشند تا مفاهیم انتزاعی سطح بالا در دادگان را مدل نمایند که این فرایند با استفاده از یک گراف عمیق که دارای چندین لایه پردازشی متشکل از چندین لایه تبدیلات خطی و غیرخطی هستند، مدل میکنند. انگیزه نخستین در به وجود آمدن این ساختار یادگیری از راه بررسی ساختار عصبی در مغز انسان الهام گرفته شده است که در آن یاخته های عصبی با فرستادن پیام به یکدیگر درک را امکان پذیر میکنند، بسته به فرض

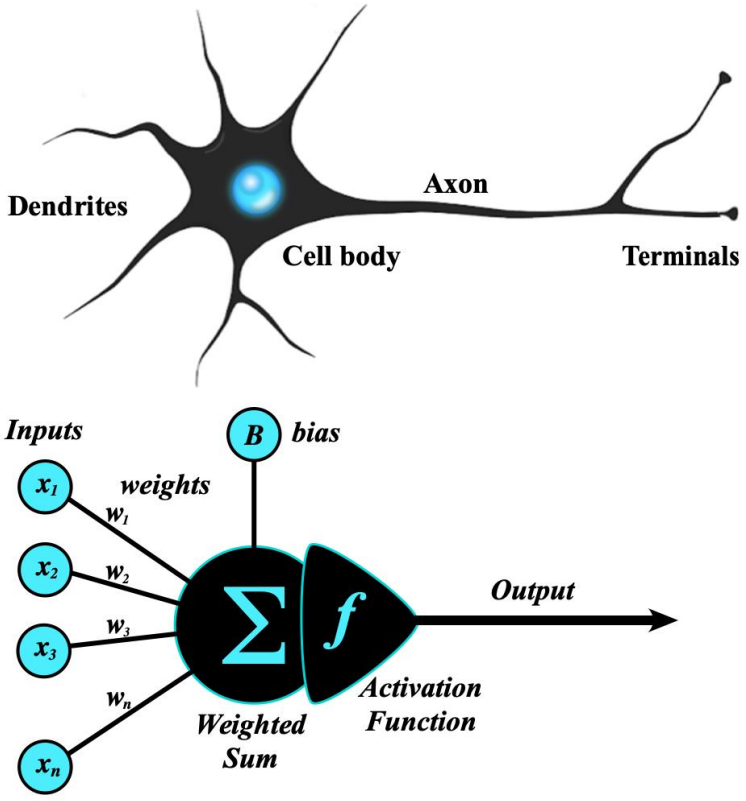
های گوناگون در مورد نحوه اتصال این یاخته های عصبی، مدل ها و ساختار های مختلفی در این حوزه پیشنهاد و بررسی شده اند، هرچند که این مدل ها به صورت طبیعی در مغز انسان وجود ندارد و مغز انسان پیچیدگی های بیشتری را داراست. از جمله این مدل ها میتوان به مدل های شبکه عصبی عمیق، شبکه عصبی کانوولوشنال و شبکه عصبی بازگشتی اشاره کرد.

۳-۲-۱ شبکه های عصبی مصنوعی

یک شبکه عصبی یا شبکه عصبی عمیق (Artificial Neural Network) از یک لایه ورودی، یک لایه خروجی و چندین لایه ی پنهان میان این دو لایه تشکیل میشود. هر لایه از تعدادی نرون تشکیل شده است.



این نرون ها از بخش های مختلفی تشکیل شده است که این بخش ها شامل تابع فعال سازی غیرخطی و وزن ها بر روی اتصالات ورودی و خروجی آن در قالب یک تبدیل خطی میشود.



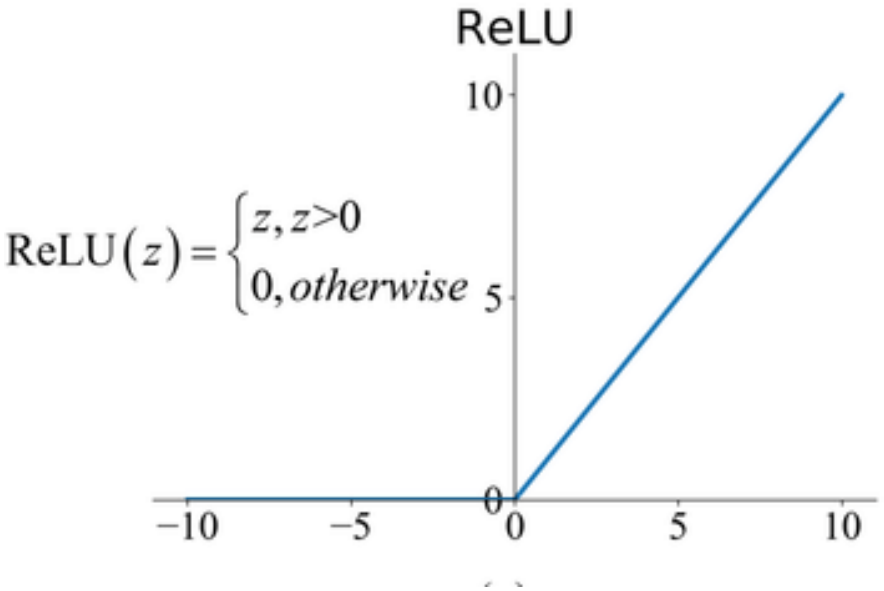
در این نرون یک بردار از وزن ها و یک مقدار bias نگهداری میشود که در نهایت پس از اعمال آنها روی ورودی های نرون، حاصل ها با یکدیگر جمع میشوند. این وزن ها اهمیت هر ورودی را نشان میدهند و تاثیر هر یک را روی خروجی این نرون مشخص میکنند. در پروسه ی آموزش شبکه عصبی مقادیر این وزن ها و bias تنظیم میشوند تا مدل به خروجی بهینه و مورد نظر برسد.

در فرمول زیر محاسبات انجام گرفته را مشاهده میکنید که خروجی این فرمول به عنوان ورودی به تابع فعالسازی داده میشود. در این فرمول یک مجموع وزن دار روی ورودی های نرون صورت میگیرد. این مجموع در نهایت با مقدار bias جمع میشود.

$$\sum_{j=1}^{n} x_j w_j$$

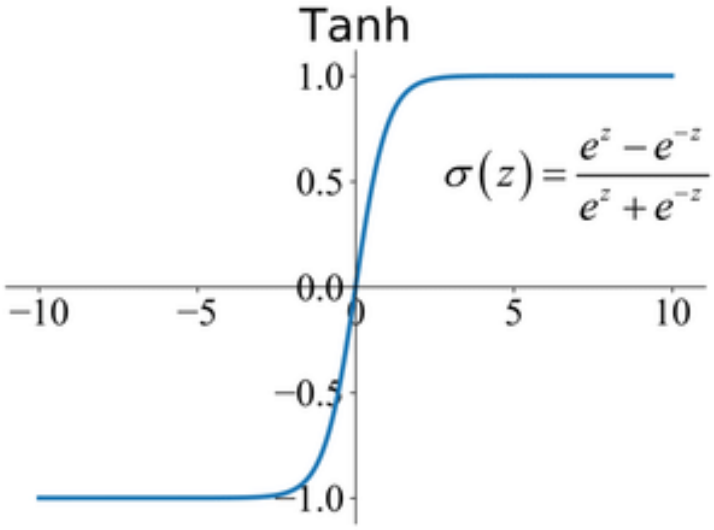
در ادامه به بررسی توابع فعالسازی میپردازیم. استفاده از این توابع غیرخطی برای بازنمایی های پیچیده و الگو های پیچیده داده ها لازم است چراکه بدون استفاده از این تابع ها تمامی نرون ها یک تبدیل خطی روی داده ها اعمال میکنند و کل شبکه به یک ترکیب خطی از ورودی ها تبدیل میشود و عملا تنها الگوهای ساده و خطی قابل استخراج میشوند. این توابع مانند نرون های مغز فعال میشوند و اتصال خود را با نرون بعدی برقرار میکنند. از توابع فعالسازی متداول که در شبکه های عصبی مورد استفاده قرار میگیرند میتوان به ReLu، leaky ReLu، tanh، sigmoid، softmax اشاره کرد که در ادامه به معرفی هر یک از آنها میپردازیم.

- **تابع فعالسازی ReLU** – تابع Rectified Linear Unit activation function یا به اختصار ReLU تابعی ست با نمودار و فرمول زیر:



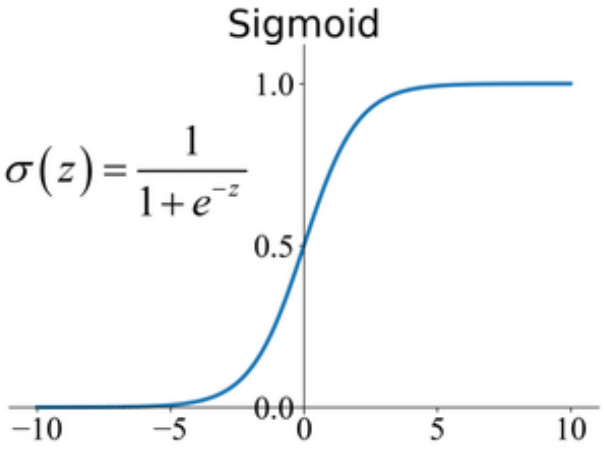
که خروجی آن برای ورودی های بزرگتر از ۰ برابر با همان ورودی است. این تابع متداول ترین تابع در شبکه های عصبی میباشد چراکه فرایند آموزش با این تابع سریعتر هست. این تابع از نظر محاسباتی نیز بسیار کارامد است و سرعت همگرایی بالایی دارد.این تابع مشکل مرگ نرون را دارد که در مقادیر ورودی نزدیک صفر گرادیان تابع صفر میشود و در آموزش تاثیر میگذارد.

- **تابع فعالسازی tanh –** تابع تانژانت هایپربولیک تابعی است با فرمول و نمودار زیر:



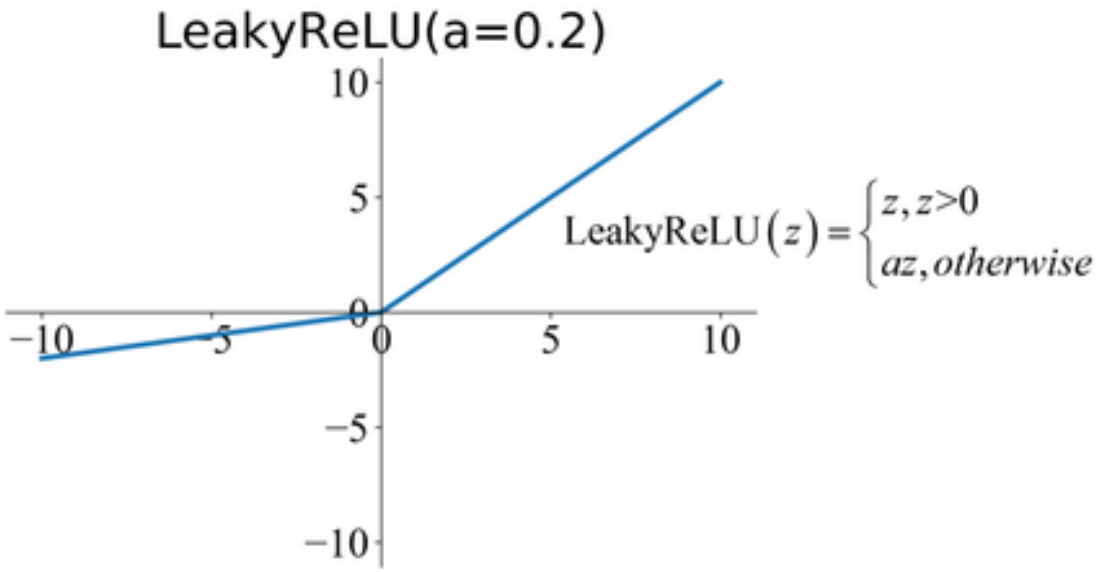
که در آن خروجی بین ۱ و 1- بسته به مقدار ورودی تغییر میکند. این تابع مشکل محوشدگی گرادیان را دارد به این معنی که در مقادیر بسیار بزرگ یا کوچک ورودی، مشتق بسیار کوچک میشود و به آموزش شبکه لطمه میزند. این تابع به همین دلیل به کندی همگرا میشود و فرایند آموزش شبکه را طولانی میکند.

- **تابع فعالسازی Sigmoid –** تابعی با فرمول و نمودار زیر:



این تابع یک منحنی S شکل است. زمانی‌که می‌خواهیم خروجی مدل احتمال باشد، از تابع سیگموید استفاده می‌کنیم. چون تابع سیگموید مقادیر را به بازه صفر تا ۱ می‌برد و احتمالات هم میان همین بازه قرار دارند. این تابع هم مشکل محوشدگی گرادیان و هم گرایی کند را دارد.

- **تابع leaky ReLU** – تابعی با فرمول و نمودار زیر:



انتخاب تابع فعالسازی برای نرون ها به نوع مسئله مرتبط میباشد و هر کدام را میتوان با توجه به شرایط استفاده کرد. نکات زیر معمولا را میتوان در رابطه با این توابع فعالسازی بیان کرد:

- تابع سیگموید در مسائل کلاس بندی معمولا به خوبی عمل میکند.
- توابع سیگموید و tanh به دلیل مشکل محوشدگی گرادیان، در برخی موارد استفاده نمیشوند.

- در بیشتر مواقع از تابع ReLU استفاده میشود که این تابع نتایج خوبی را ارائه میکند.
- تابع ReLU تنها در لایه های پنهان مورد استفاده قرار میگیرد چرا که مقدار آن تا بینهایت میرود و ما در خروجی معمولا به یک احتمال یا یک مقدار کراندار نیاز داریم.
- در مواجهه با مشکل مرگ نرون میتوان از تابع leaky ReLU استفاده کرد.

فرایند آموزش شبکه عصبی

فرایند آموزش یا training process به تنظیم کردن وزن های هر نرون برای مینیمم کردن خطای مدل گفته میشود. در این فرایند با استفاده از یک تابع خطا، خطای مدل در پیشبینی خروجی با توجه به مجموعه ورودی های مدل (مجموعه داده های آموزش یا training data) محاسبه میشود و با توجه به این خطا وزن ها تنظیم میشوند. یکی از روش هایی که برای اینکار وجود دارد روش پس انتشار خطا یا back propagation of error نام دارد. در این روش دو مرحله وجود دارد که در مرحله ی اول که حرکت رو به جلو یا feed forward نام دارد، داده های ورودی از لایه ی ورودی به شبکه داده میشوند و این داده ها به سمت لایه خروجی رفته و عملیات هر نرون روی آنها اعمال میشود (مجموع وزن دار و توابع فعالسازی). سرانجام پس از این مرحله یک خروجی بدست میاید که این خروجی مقداری خطا نسبت به

خروجی واقعی دارد. اینجا مقدار خطا توسط تابع خطا محاسبه میشود و وارد مرحله دوم میشود. در این مرحله که انتشار رو به عقب یا back propagation نام دارد، وزن ها به هنگام سازی میشوند. این به هنگام سازی با توجه به مقدار تابع خطا صورت میگیرد تا در تکرار های بعدی این عملیات خطای کمتری داشته باشیم. در نهایت هدف رسیدن به کمترین مقدار خطا میباشد. در مرحله دوم این روش از مقدار خطا شروع به بازگشت به عقب میکنیم تا به لایه ورودی برسیم. در این مسیر مقدار گرادیان ها را محاسبه میکنیم و با داشتن مقدار خطا، سعی در مینیمم کردن مقدار خطا میکنیم. برای اینکار میتوان از روش کاهش گرادیان یا Gradient Descent استفاده کرد که در این روش با محاسبه ی گرادیان تابع خطا وزن ها را بهینه سازی میکند تا مقدار تابع خطا به مقدار مینیمم آن نزدیک شود.

توابع خطا یا توابع زیان (loss function) مختلفی برای استفاده در شبکه های عصبی وجود دارند که میتوان چند نمونه از آن ها را در ادامه مشاهده کرد (در فرمول هایی که در ادامه آورده شده است، $\hat{y}$ نشان دهنده ی مقدار پیشبینی شده توسط شبکه و y نشان دهنده ی مقدار واقعی خروجی میباشد)

- **تابع Means Square Error** – یکی از متداول ترین توابع مورد استفاده در شبکه های عصبی که برای مسائل رگرسیون هستند. این تابع میانگین مربعات خطا نسبت به مقدار واقعی را محاسبه میکند. فرمول این تابع به صورت زیر میباشد:

$$MSE = \frac{\sum(y_i - \hat{y}_i)^2}{n}$$

- تابع **Mean Absolute Error** – از این تابع نیز در مسائل رگرسیون استفاده میشود. این تابع میانگین قدرمطلق تفاضل بین مقدار پیشبینی شده و مقدار واقعی را محاسبه میکند. فرمول این تابع به صورت زیر میباشد:

$$MAE = \frac{\sum |y_i - \hat{y}_i|}{n}$$

- تابع **Binary Cross-entropy** – از این تابع در مسائل کلاس بندی با دو کلاس استفاده میشود. فرمول این تابع به صورت زیر میباشد:

$$-\frac{1}{\text{output size}} \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

در نهایت با استفاده از این توابع مقدار کلی خطا برای تمامی داده های آموزش با میانگین گیری روی تمام خطا ها محاسبه میشود.

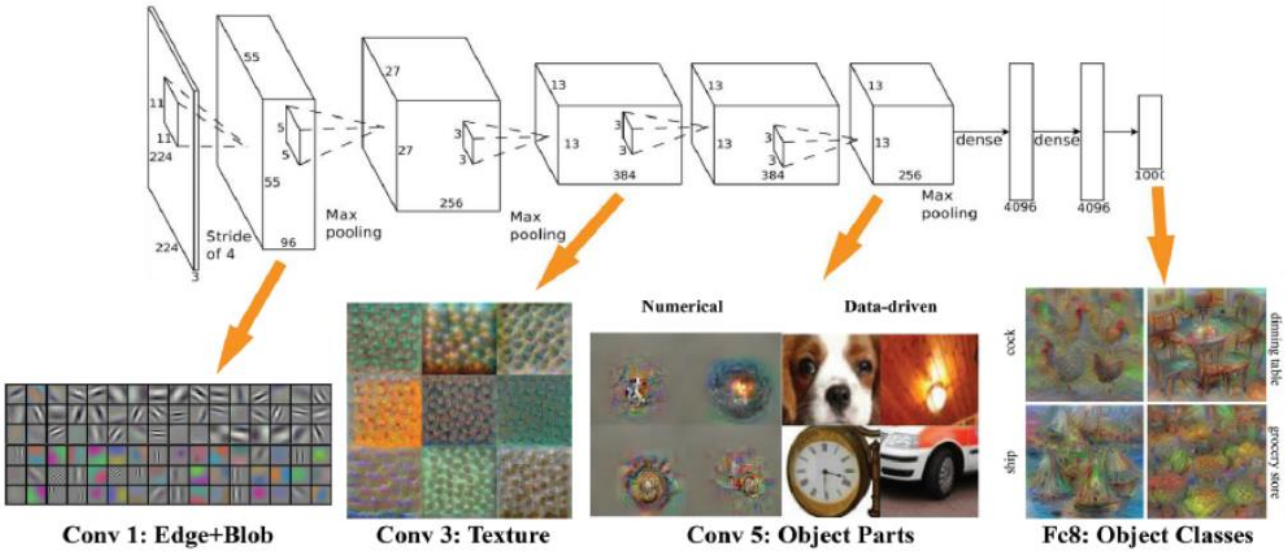
در نهایت با استفاده از روش گرادیان کاهشی و تکرار بر روی داده های آموزش، مقادیر وزن ها برای مینیمم سازی میزان خطا بهینه میشوند تا مدل با خطای کمی خروجی را پیشبینی کند.

۳-۲-۲ شبکه های عصبی کانوولوشنال

یکی از مدل های رایج شبکه های عصبی عمیق، شبکه عصبی کانوولوشنال یا Convolutional Neural Network یا به اختصار CNN میباشد. این نوع شبکه عصبی برای پردازش تصویر بسیار مناسب میباشد و این شبکه ها یک بازنمایی پیچیده از داده های تصویری را با استفاده از مقدار حجیمی از داده ها یاد میگیرند. این شبکه ها از سیستم بینایی انسان الهام گرفته شده اند و چندین لایه از تبدیلات را یاد میگیرند. این شبکه ها از دو نوع لایه تشکیل میشوند که به شرح زیر میباشند:

- **لایه کانوولوشن** – این لایه با توجه به اینکه در تصاویر پیکسل ها در کنار هم معنا دار هستند و تک به تک معنایی ندارند و یک locality بین آنها برقرار است، با عملیات کانوولوشن بر روی آنها، ویژگی ها را استخراج میکند.

- **لایه تلفیق** – در این لایه ماکزیمم یا متوسط ویژگی ها بر روی یک ناحیه خاص محاسبه میشوند تا اندازه ی تصویر کاهش یابد و محاسبات در لایه های بعدی سبک تر شود.

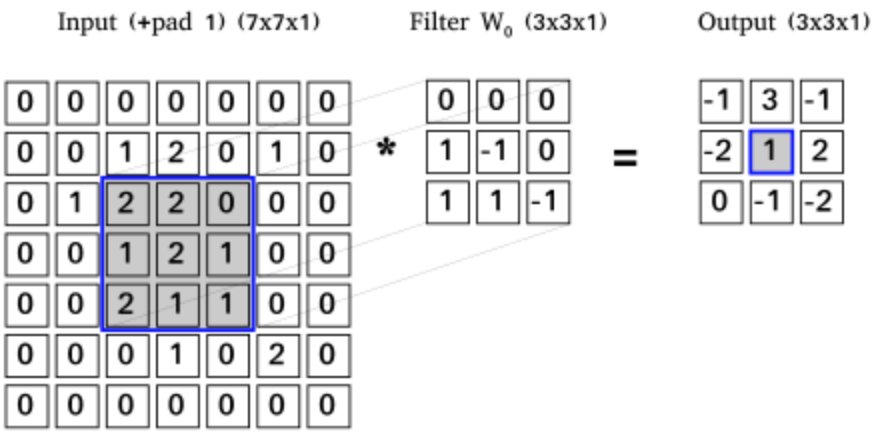


هر لایه ی کانوولوشن از تعدادی فیلتر تشکیل شده است که هر یک از این فیلتر ها به صورت یک ماتریس دوبعدی از اعداد هستند که در اصل معادل با وزن ها در شبکه عصبی مصنوعی عمل میکنند و در نهایت هدف تنظیم این مقادیر برای بهینه سازی عملکرد شبکه میباشد.

عملیات کانوولوشن:

در این عملیات یک ماتریس دوبعدی به عنوان فیلتر بر روی مقادیر پیکسل های تصویر (که خود نیز یک ماتریس دوبعدی میباشد) اعمال میشود. این عملیات سبب میشود تا به جای یادگیری یک وزن به ازای هر پیکسل از

تصویر، با یادگیری وزن های فیلتر به عملکردی بهتر و بهینه تر از نظر محاسباتی رسید. در تصویر زیر میتوان نحوه ی عملکرد این عملیات را مشاهده کرد:



در تصویر بالا مربع بزرگ مقادیر تصویر و مربع کوچک وسطی مقادیر فیلتر را نشان میدهند. در هر مرحله فیلتر بر روی بخشی از تصویر اعمال میشود و خروجی آن بخش که یک عدد میباشد را در ماتریس خروجی قرار میدهد. این فیلتر با یک گام مشخص (برای مثال دو پیکسل دو پیکسل) روی تصویر حرکت میکند تا تمامی بخش های تصویر را پوشش دهد. نحوه ی محاسبات به این صورت میباشد که هر درایه از ماتریس در درایه متناظر با آن در فیلتر ضرب میشود و در نهایت نتیجه ی تمامی این ها با هم جمع شده و به عنوان خروجی در نظر گرفته میشود. برای مثال برای تصویر بالا داریم:

$$2x0 + 2x0 + 0x0 + 1x1 + 2x-1 + 1x0 + 2x1 + 1x1 + 1x-1 = 1$$

پس از این که خروجی عملیات کانوولوشن در لایه کانوولوشن مشخص شد این ماتریس با ماتریس bias جمع شده و نتیجه آن به یک تابع فعالسازی داده میشود و خروجی این تابع فعالسازی به عنوان ورودی لایه بعدی در نظر گرفته میشود.

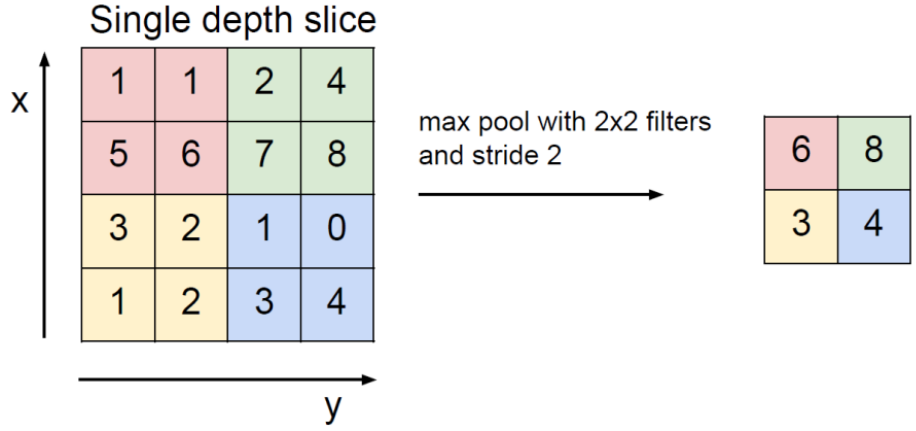
عملیات تلفیق:

از این عملیات برای کاهش ابعاد بازنمایی ها استفاده میشود. روش های مختلفی برای این عملیات وجود دارد که در ادامه به آنها میپردازیم.

- Max-Pooling – در این روش هر فیلتر با یک گام مشخص بر روی تصویر حرکت میکند و هر بار مقدار ماکزیمم را به عنوان خروجی باز میگرداند. فرمول آن به صورت زیر میباشد:

$$h_i^n(r,c) = \max_{\bar{r}\in N(r),\ \bar{c}\in N(c)} h_i^{n-1}(\bar{r},\bar{c})$$

در تصویر زیر میتوان اعمال این عملیات بر یک ماتریس را مشاهده کرد:

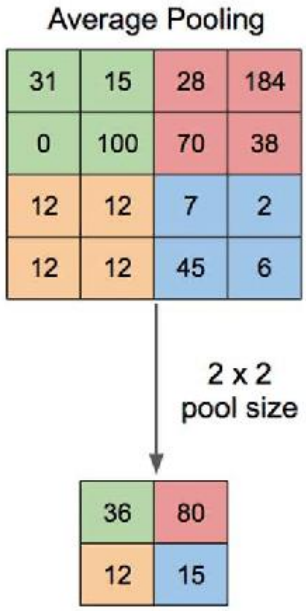


در تصویر بالا یک فیلتر max-pool ۲ در ۲ با اندازه گام ۲ بر روی تصویر اعمال شده که خروجی به صورت ماتریس سمت راست میشود.

- Average-Pooling – در این روش هر فیلتر با یک گام مشخص بر روی تصویر حرکت میکند و هر بار مقدار میانگین را به عنوان خروجی باز میگرداند. فرمول آن به صورت زیر میباشد:

$$h_i^n(r,c) = \underset{\bar{r}\in N(r),\ \bar{c}\in N(c)}{\text{mean}} h_i^{n-1}(\bar{r},\bar{c})$$

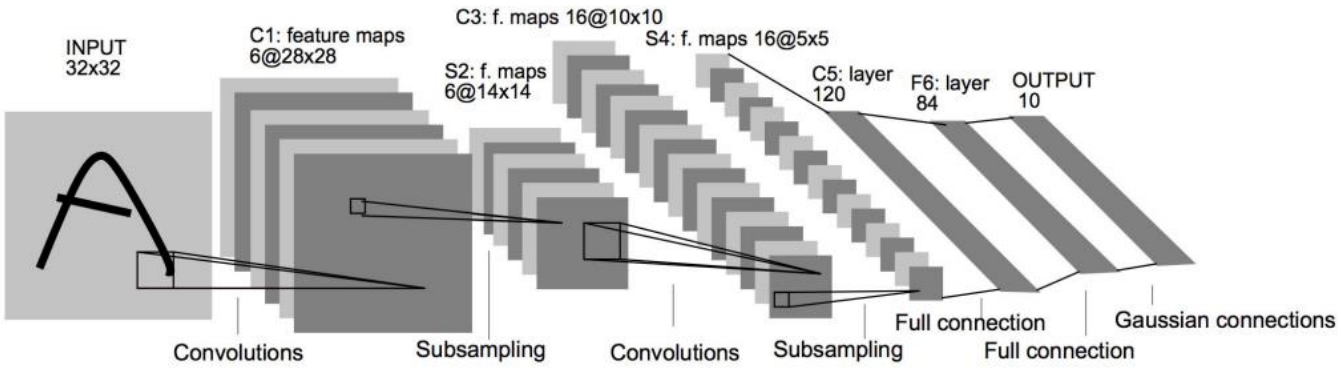
در تصویر زیر میتوان اعمال این عملیات بر روی یک ماتریس مشاهده کرد:



در تصویر بالا یک فیلتر average-pool ۲ در ۲ با اندازه گام ۲ بر روی تصویر اعمال شده که خروجی به صورت ماتریس پایینی میشود.

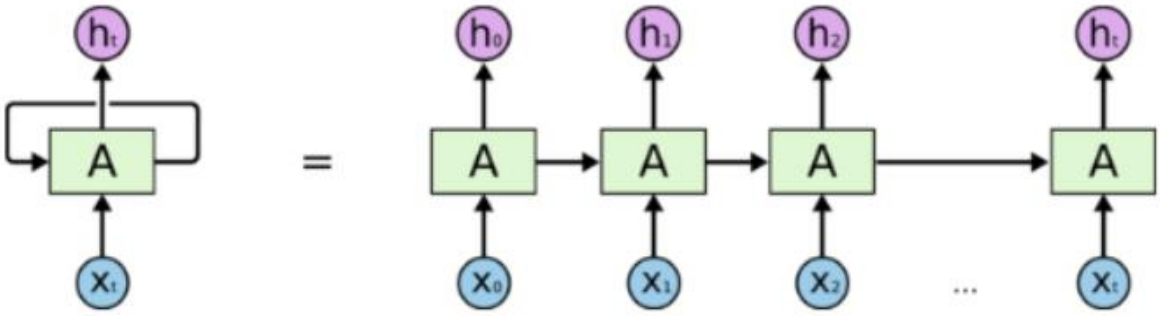
این دو روش از متداول ترین روش ها میباشند و معمولا از یکی از این دو روش در شبکه های کانوولوشنال استفاده میشود.

پس از لایه های کانوولوشن و تلفیق، برای بدست آوردن خروجی که به صورت تک عدد میباشد، نمیتوان از فیلتر ها که خروجیشان ماتریسی از اعداد است استفاده کرد و در لایه های اخر شبکه های کانوولوشنال از یک لایه ی تماما متصل (Fully Connected Layer) استفاده میشود. ورودی این لایه تخت شده ی ماتریس خروجی اخرین لایه ی کانوولوشن یا تلفیقی (بسته به معماری شبکه) میباشد. به این معنا که ماتریس دوبعدی خروجی لایه قبل به یک آرایه تک بعدی تبدیل میشود و این آرایه به عنوان ورودی به این لایه داده میشود. این لایه همانند لایه پنهان در شبکه عصبی مصنوعی میباشد و هر نرون آن یک وزن و یک bias دارد. در شبکه های کانوولوشنال معمولا چندین لایه ی آخر شبکه لایه های Fully Connected میباشند که در لایه ی خروجی با توجه به نوع مسئله تابع فعالسازی مناسب انتخاب میشود. در تصویر زیر یک نمونه از شبکه عصبی کانوولوشنال را مشاهده میکنید:



۳-۲-۳ شبکه های عصبی بازگشتی

شبکه عصبی بازگشتی Recurrent Neural Network یا به اختصار RNN نوعی از شبکه های عصبی میباشند که در تشخیص گفتار، پردازش زبان طبیعی، پردازش صوت و به صورت کلی پردازش داده های ترتیبی (Sequential data) استفاده میشود. در این شبکه ها برخلاف شبکه های معرفی شده تا اینجا، یک لایه ی بازخورد داریم که خروجی شبکه به همراه ورودی بعدی به شبکه بازگرداننده میشود. ای شبکه ها با استفاده از این مکانیزم دارای حافظه داخلی میباشند و میتوانند ورودی های قبلی را به خاطر بسپارند و از این حافظه برای پردازش دنباله ای از ورودی ها بهره ببرند.

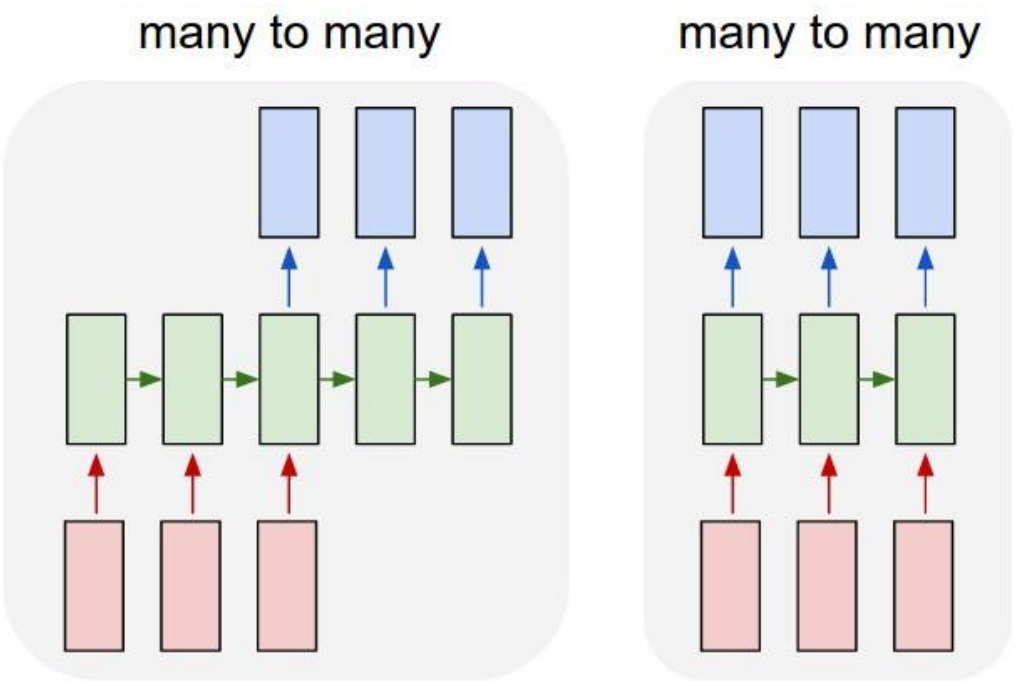


An unrolled recurrent neural network.

در تصویر بالا یم لایه از شبکه عصبی بازگشتی را مشاهده میکنید که در اصل یک واحد میباشد که از خروجی زمان t-1 در زمان t به عنوان ورودی استفاده میکند و با استفاده از مکانیزم هایی میتواند اطلاعاتی از زمان های قبل تر هم نگه دارد.

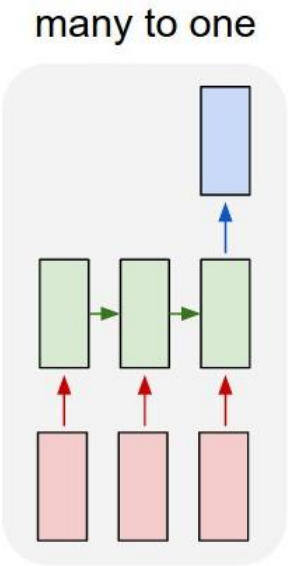
معماری های مختلفی از این شبکه وجود دارد که میتوان به نمونه های زیر اشاره کرد:

- Many-to-many در این معماری هم ورودی و هم خروجی یک دنباله میباشند. از موارد استفاده این معماری میتوان به ترجمه ماشینی اشاره کرد.

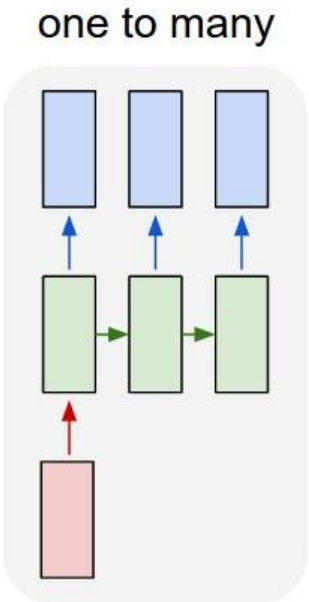


- Many-to-one در این معماری ورودی به صورت یک دنباله و خروجی به صورت یک عدد میباشد. از موارد استفاده ی این معماری میتوان به تحلیل احساسات اشاره کرد که با گرفتن یک جمله که دنباله ای از کلمات است، یک عدد در رابطه با احساس جمله ی ورودی (مثبت یا

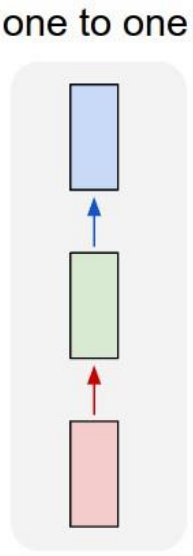
منفی بودن آن) بازمیگرداند. از دیگر موارد استفاده این معماری که در این پروژه نیز استفاده شده به دسته بندی صوت میتوان اشاره کرد.



- One-to-many در این معماری ورودی به صورت یک عدد و خروجی یک دنباله میباشد. از موارد استفاده این معماری میتوان به تولید موسیقی با ورودی ژانر آن اشاره کرد.

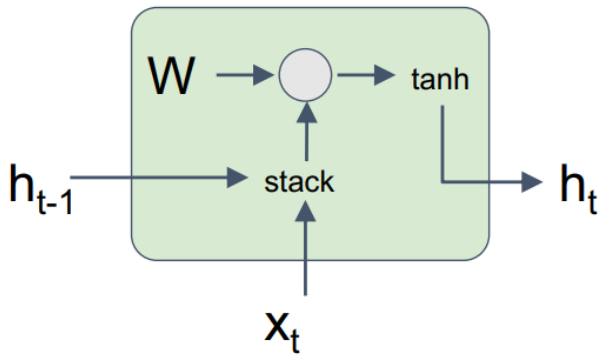


- One-to-One در این معماری هم ورودی و هم خروجی یک عدد میباشند. این معماری همانند یک شبکه عصبی مصنوعی عمل میکند.



در شبکه های عصبی بازگشتی نیز هر نرون حاوی وزن ها و bias و یک تابع فعالسازی میباشد و تنها معماری آن متفاوت میباشد. در این مدل شبکه عصبی معمولا از تابع فعالسازی tanh استفاده میشود.

یک واحد ساده از شبکه عصبی بازگشتی به صورت زیر میباشد:



فرمول های مورد استفاده در این واحد به صورت زیر میباشد:

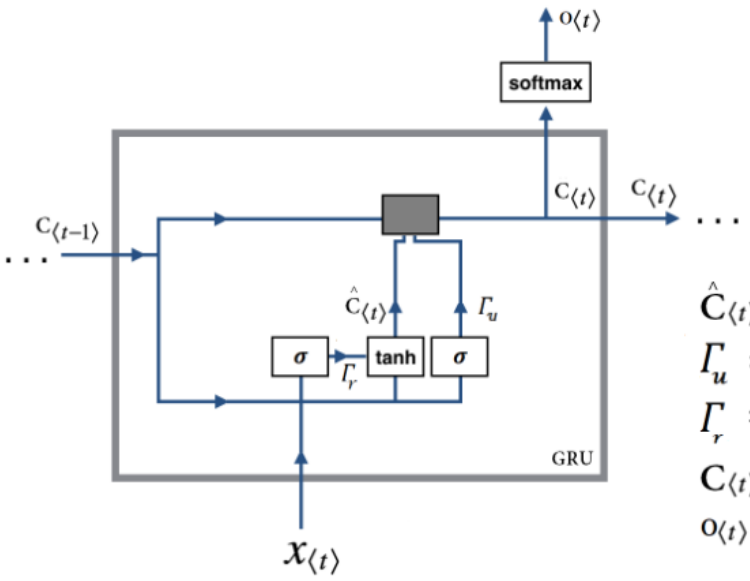
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

که در این فرمول W_{hh} نشان دهنده ی وزن های اعمال شده روی خروجی واحد در زمان قبلی (h_{t-1}) و W_{xh} نشان دهنده وزن های اعمال شده روی ورودی در این زمان (x_t) میباشد. در نهایت پس از اعمال وزن ها از تابع فعالسازی میگذرند و خروجی این واحد را در زمان t تشکیل میدهند.

این مدل ساده تنها از خروجی یک زمان قبل تر استفاده میکند و از مشکل محو شدگی گرادیان رنج میبرند. برای استفاده از اطلاعات زمان های قبل تر باید به این واحد حافظه اضافه کنیم. دو مدل واحد متعارف حافظه دار با نام های LSTM و GRU وجود دارند.

GRU یا Gated Recurrent Unit به استفاده از گیت ها برای نگهداری اطلاعات از گام های زمانی قبلی میباشد. در این واحد گیت های بروزرسانی و بازنشانی (Update gate و Reset gate) که مشخص میکنند چه اطلاعاتی به خروجی منتقل شوند و چه اطلاعاتی منتقل نشوند.

در تصویر زیر میتوان یک واحد GRU و فرمول های مربوط به گیت های آن را مشاهده کرد:

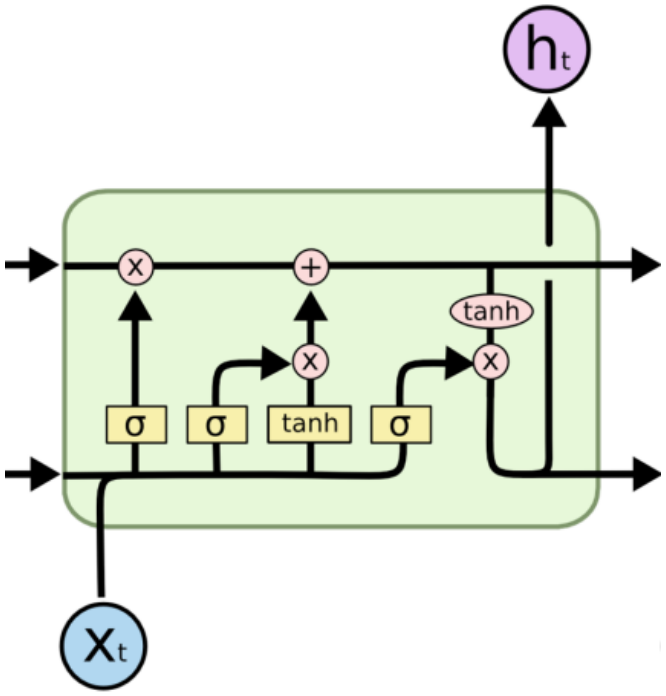


$$\hat{C}_{\langle t\rangle} = \tanh(W_1[\Gamma_r * C_{\langle t-1\rangle}, x_{\langle t\rangle}] + b_c)$$
$$\Gamma_u = \sigma(W_2[C_{\langle t-1\rangle}, x_{\langle t\rangle}] + b_u)$$
$$\Gamma_r = \sigma(W_3[C_{\langle t-1\rangle}, x_{\langle t\rangle}] + b_u)$$
$$C_{\langle t\rangle} = \Gamma_u * \hat{C}_{\langle t\rangle} + (1 - \Gamma_u) * C_{\langle t-1\rangle}$$
$$o_{\langle t\rangle} = soft\max(W_4 C_{<t>} + b_o)$$

در اینجا C همان سلول حافظه را نشان میدهد که در اصل همان h_t در واحد ساده میباشد. فرمول مربوط به گیت بروزرسانی با Γ_u نشان داده میشود و فرمول مربوط به گیت بازنشانی با Γ_r نمایش داده میشود. هر یک از این گیت ها وزن های مخصوص به خود را دارند که همگی در فرایند آموزش تنظیم میشوند.

LSTM یا Long Short Term Memory دیگر واحد حافظه دار برای شبکه های بازگشتی میباشد. این واحد هم مانند GRU با استفاده از گیت ها اطلاعات را از زمان های قبلی به خاطر میسپارد. در این واحد یک گیت جدید به نام گیت فراموشی (Forget Gate) وجود دارد که در صورت نیاز به

فراموش کردن اطلاعات قبلی منجر میشود. در تصویر زیر میتوان شکل کلی این واحد و فرمول های مربوط به آن را مشاهده کرد:



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

$$o_t = \sigma\left(W_o \; [\, h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

در فرمول های بالا فرمول اول مربوط به گیت فراموشی و فرمول دوم مربوط به گیت ورودی و فرمول سوم مربوط به حافظه ی داخلی و فرمول چهارم به گیت خروجی و فرمول آخر مربوط به خروجی واحد میباشد. در این پروژه از این واحد در شبکه عصبی بازگشتی استفاده شده است بدلیل ماهیت مسئله و اینکه برای تشخیص یک کلمه در یک سیگنال صوتی اطلاعات گام های زمانی قبلی در تصمیم گیری تاثیر دارند.

۳-۳ ابزارها

در این بخش به معرفی ابزار های مورد استفاده برای پیاده سازی این پروژه میپردازیم.

برای پیاده سازی این پروژه در تمامی برنامه ها از زبان برنامه نویسی پایتون استفاده شده است. در ادامه کتابخانه های کمکی که استفاده شده اند را معرفی میکنیم.

- **کتابخانه numpy:** یک کتابخانه پایتون است که برای کار با آرایه ها به وجود آمده است. کتابخانه نامپای همچنین توابعی برای انجام عملیات های گوناگون در جبرخطی، تبدیل فوریه و ماتریس ها دارد. نامپای در سال ۲۰۰۵ و به صورت متن باز ایجاد شد. بخش زیادی از این کتابخانه به دلیل سرعت پردازش با زبان c و c++ پیاده سازی شده است.

- **کتابخانه librosa**: یک کتابخانه پایتون که برای کار با داده های صوتی میباشد و امکاناتی نظیر خواندن فایل های صوتی، استخراج ویژگی ها از سیگنال های صوتی، اعمال تغییرات و تبدیل های متداول بر روی سیگنال های صوتی، تصویرسازی انواع بازنمایی های صوت، ذخیره سازی خروجی فایل صوتی و .... دارد.
- **کتابخانه Keras**: یک کتابخانه رایگان متن باز با کاربرد آسان برای توسعه و ارزیابی مدل های یادگیری عمیق است. این کتابخانه به ما این امکان را میدهد که فقط در چند خط کد مدل های شبکه عصبی را پیاده سازی کنیم. این کتابخانه شامل پیاده سازی انواع مختلف اجزای شبکه های عصبی از جمله لایه های مختلف، واحد های مختلف، معماری های مختلف و الگوریتم های بهینه سازی مختلف میباشد که کار با آنها بسیار ساده است.

در فصل های بعدی به توضیح بخش های مختلف پیاده سازی شده و توضیح کد های آنها خواهیم پرداخت.

فصل چهارم

۴-۱ جمع آوری داده ها

در این فصل به جمع آوری داده ها و آماده سازی آنها برای استفاده مدل یادگیری عمیق میپردازیم.

مدل های یادگیری عمیق برای پیدا کردن الگوها و یادگیری آنها برای پیشبینی کردن خروجی مدنظر، به تعداد زیادی داده نیاز دارند. در این پروژه این داده ها سیگنال های صوتی میباشند که باید آنها را به دو دسته ی کلمه کلیدی (true class) و کلمه غیرکلیدی (false class) تقسیم کرد. برای مجموعه دادگان این پروژه از دیتاست فارسی موجود در لینک زیر استفاده شده است:

https://commonvoice.mozilla.org/en/datasets

این مجموعه دادگان که Common Voice نام دارد، یک پروژه جمع سپاری است که همه افراد با ضبط صدای خود و بازگو کردن جملات میتوانند به این مجموعه دادگان داده اضافه کنند. این مجموعه به صورت رایگان میباشد و در پروژه های تشخیص گفتار کاربرد دارد. این مجموعه دادگان شامل دو بخش فایل های صوتی و یک فایل که محتوای هر فایل صوتی و جملاتی که در هر فایل گفته شده، میباشد. دیتاست فارسی این مجموعه از حدود ۳۰۰ ساعت

صوت تشکیل شده و شامل بالغ بر ۴۰۰۰ گوینده میباشد. فایل های صوتی داخل این دیتاست با فرمت mp3 هستند. برای این پروژه از آخرین بروزرسانی این دیتاست یعنی Common Voice Corpus 10.0 استفاده شده است. در بخش های بعدی این فصل به نحوه ی استخراج تک کلمات از این دیتاست خواهیم پرداخت.

۴-۱-۱ جمع آوری دستی داده ها

پس از بررسی دیتاست Common Voice با مشکل کمبود دیتا برای کلمه کلیدی مواجه شدیم. برای رفع این مشکل با پیاده سازی یک کد برای ضبط صدا از میکروفون، تعداد داده ها برای کلمه ی کلیدی را افزایش دادیم. از این کد برای ضبط صدای محیط و نویز نیز استفاده کردیم چراکه بعد از آموزش مدل و تست کردن آن متوجه این موضوع شدیم که بدلیل عدم وجود نویز و صدای محیط در دیتاست، مدل به اشتباه نویز و صدای سکوت و محیط را به عنوان کلمه کلیدی تشخیص میداد که با اضافه کردن این داده ها به مجموعه داده های آموزش مدل این مشکل برطرف شد.

در ادامه به توضیح برنامه ی record.py که به همین منظور پیاده سازی شد میپردازیم.

```
import sounddevice as sd
from scipy.io.wavfile import write
import keyboard as kb
import os
```

ابتدا کتابخانه های استفاده شده در این کد را اضافه میکنیم. در این کد از کتابخانه ی sounddevice برای گرفتن ورودی از میکروفون استفاده شده است. از کتابخانه ی scipy برای ذخیره کردن صوت ضبط شده استفاده شده و از کتابخانه ی keyboard برای گرفتن ورودی از کیبورد استفاده شده است.

```
def record(num,sr=22050,seconds=1):
    '''
    Parameters
    ----------
    num : int
        number of audio files to be recorded in this function call.
    sr : int, optional
        recording sample rate. The default is 22050.
    seconds : int, optional
        duration of each recording. The default is 1.

    Returns
    -------
    saves recorded files in the same directory as the code.
    '''
```

در این کد از تابع record برای ضبط و ذخیره صوت از طریق میکروفون استفاده شده است. ورودی های این تابع بصورت زیر میباشند:

- num: یک عدد صحیح که تعداد فایل هایی که با هر بار صدا زدن باید ضبط شوند را مشخص میکند.
- sr: samplerate که نرخ نمونه برداری سیگنال صوتی را نشان میدهد. در این پروژه در تمامی بخش ها از نرخ نمونه برداری ۲۲۰۵۰ استفاده شده است. به این معنا که هر ۲۲۰۵۰ نمونه، یک ثانیه صوت را تشکیل میدهند. مقدار پیش فرض این آرگومان در این تابع نیز ۲۲۰۵۰ قرار داده شده است.
- seconds: طول فایل های صوتی ضبط شده توسط تابع را نشان میدهد. به صورت پیشفرض مقدار آن برابر با ۱ قرار داده شده است.

خروجی این تابع ذخیره فایل های صوتی ضبط شده به تعداد آورده شده در ورودی آن میباشد که این فایل ها در همان فولدری که کد در آن قرار دارد ذخیره میشوند.

نحوه نامگذاری فایل های ذخیره شده توسط این تابع به صورت ‘keyword_n.wav’ میباشد که در آن n نشان دهنده شماره ی فایل ضبط شده است.

```
list1 = os.listdir(os.getcwd())
c = 0
for i in list1:
    if i.split('.')[0][0:7] == 'keyword' and int(i.split('.')[0][8]) > c:
        c = int(i.split('.')[0][8])
```

در این بخش از کد لیست فایل های موجود در فولدر کد بررسی میشود تا اگر

از اجرا های قبلی فایلی با فرمت نام گذاری این برنامه وجود دارد، شماره ی فایل ها برای این اجرا از یک شماره بعد از فایل های از قبل موجود باشد. اینجا متغیر c شماره آخرین فایل از قبل موجود را (در صورت وجود) در خود نگه میدارد.

```
c += 1
for i in range(num):
    while True:
        print('press "S" to start recording:')
        if kb.read_key() == 's':
            print(str(seconds) + ' second started.')
            myrecording = sd.rec(int(seconds * sr), samplerate=sr, channels=2)
            sd.wait()
            write('keyword_' + str(c)+'.wav', sr, myrecording)
            print('recorded file number '+str(c))
            print('------------------------------------------------------------')
            c += 1
            break
```

در این قسمت از تابع یک حلقه به تعداد num داریم که در هر دور یک فایل ذخیره میشود. روش کار کلی برای ضبط صدا به این صورت میباشد که برنامه منتظر فشردن کلید s میماند و هر زمان که این کلید فشرده شد، شروع به ضبط کردن صدا از طریق میکروفون به اندازه ی seconds ثانیه میکند.

برای ضبط صدا از طریق میکروفون از تابع rec از کتابخانه ی sounddevice استفاده شده است. این تابع به عنوان ورودی تعداد نمونه های مورد نیاز برای ضبط (ورودی اول) که در این تابع این مقدار با ضرب نرخ نمونه برداری در طول هر فایل صوتی به ثانیه بدست می آید. ورودی دوم این تابع نرخ نمونه برداری را مشخص میکند و ورودی سوم هم تعداد کانال ها

برای ضبط صدا از طریق میکروفون را مشخص میکند. در ادامه با استفاده از تابع write از کتابخانه ی scipy سیگنال صوتی ضبط شده از طریق میکروفون را در یک فایل با فرمت wav. ذخیره میکند.

۲-۴ بررسی کلی داده ها

برای بررسی کلی داده ها و اطلاع از پر تعداد ترین کلمات بازگو شده در فایل های صوتی دیتاست و جداسازی فایل هایی که حاوی کلمات کلیدی هستند و انتقال آنها به فولدری مجزا کد data_prep.py پیاده سازی شد. در ادامه به بررسی کد data_prep.py میپردازیم.

```
import pandas as pd
import numpy as np
import os
import shutil
import librosa as lib
import soundfile as sf
```

ابتدا کتابخانه های مورد استفاده در این کد را به آن اضافه میکنیم. از کتابخانه ی pandas برای خواندن و کار با فایل محتوای صوت های موجود در دیتاست و کار با دیتا در فرمت جدول استفاده میشود. از کتابخانه ی shutil برای انتقال فایل ها به فولدر های مجزا استفاده شده است و از کتابخانه ی librosa برای کار با فایل های صوتی استفاده شده است. کتابخانه ی soundfile نیز برای تبدیل و ذخیره سیگنال های صوتی استفاده شده است.

```
df = pd.read_table('data2//cv-corpus-10.0-2022-07-04//fa//validated.tsv')
dic = {}

for i in df:
    a = i['sentence'].split(' ')
    for j in a:
        if j not in dic.keys():
            dic[j] = 1
        else:
            dic[j]+=1

ll = list(dic.values())
ll.sort()
for i in dic.keys():
    if dic[i] > 700:
        print(i)
```

در این قسمت از کد، فایل محتوای فایل های صوتی که در فرمت tsv یا tab separated values میباشد را با استفاده از کتابخانه ی pandas خوانده و در متغیر df ذخیره میکنیم. برای اینکار از تابع read_table استفاده شده است که محتوای فایل را در فرمت یک جدول باز میگرداند. سپس یک دیکشنری با عنوان dic ایجاد شده که بعد تر برای شمارش تعداد تکرار هر کلمه مورد استفاده قرار میگیرد. برای اینکار یک حلقه بر روی تمامی سطر های جدول df میزنیم و برای هر سطر، ستون sentence که جمله ای که در هر فایل صوتی گفته شده را مشخص میکند، این جمله را با جداسازی کلماتش در متغیر a ذخیره میکنیم و هر کلمه اش را شمارش میکنیم. در نهایت در حلقه انتهای این قسمت کلماتی که بیش از ۷۰۰ بار تکرار شده اند را در کنسول چاپ میکنیم. دلیل اینکار انتخاب کلمه کلیدی میباشد چراکه برای اینکه مدل بتواند به خوبی عمل کند و هر کلمه را در فرایند آموزش یاد

بگیرد، نیاز به تعداد زیادی داده دارد. با بررسی این دیتاست، کلمات زیر تعداد تکرار بیش از ۷۰۰ بار را داشته اند:

"کجاست، چقدر، چطور، بیشتر، بچه، ساعت، فارسی، شروع، تماس، دوباره، ایران، پرداخت "

به دلیل اینکه از ابتدای پروژه قصد تشخیص کلمه ی "سلام" را داشتیم، تعداد این کلمه را نیز در دیتاست چک کردیم که این تعداد برابر با ۳۰۰ تکرار بود.

```
#-------- extract filenames that contain key words
df = pd.read_table('data2//cv-corpus-10.0-2022-07-04//fa//validated.tsv')
ll = []
for i in range(len(df)):
    l = df.iloc[i]['sentence'].split(' ')
    if 'سلام' in l:
        ll.append(df.iloc[i]['path'])
ll = pd.DataFrame(ll)
ll.to_csv('سلام.csv')
```

در این قسمت از کد، اسامی فایل هایی که حاوی کلمه سلام هستند را جدا میکنیم. برای اینکار مانند بخش قبلی فایل محتوا را در قالب جدول خوانده و ذخیره میکنیم. سپس با چک کردن جملات هر فایل، نام فایلی که در جمله آن کلمه ی سلام گفته شده است را از ستون path میخوانیم و در یک لیست با نام ll آن را ذخیره میکنیم. در انتهای کار نیز این لیست را با فرمت csv در فولدری که کد در آن قرار دارد ذخیره میکنیم تا بعدتر با استفاده از آن، فایل هایی که حاوی کلمه کلیدی هستند را جدا کنیم.

```
#--------- move validated datas to a seperate directory

list1 = os.listdir('data2//cv-corpus-10.0-2022-07-04//fa//clips')
df =  pd.read_table('data2//cv-corpus-10.0-2022-07-04//fa//validated.tsv')['path']
df = df.values.tolist()

for i in list1:
    if i in df:
        shutil.move(
'E://project//data2//cv-corpus-10.0-2022-07-04//fa//clips//'+i,'E://project//data2//validated')
```

دیتاست Common Voice حاوی دو نوع دیتا میباشد. نوع اول validated است که برای آنها رونوشت وجود دارد (جمله ی گفته شده در فایل صوتی به صورت متن در فایل محتوا آورده شده است) و یک نوع هم invalidated که رونوشت آن موجود نیست. اما تمامی فایل های صوتی این دیتاست پس از دانلود در یک فولدر میباشند و برای جداسازی فایل های دارای رونوشت کد بالا پیاده شده است. در اینجا از تابع listdir از کتابخانه os که آدرس فولدر را بعنوان ورودی دریافت میکند و لیستی از اسامی فایل های داخل آن فولدر را بازمیگرداند، استفاده شده است.

```
# seperate true class
list1 = os.listdir('data2//validated')
df = pd.read_csv('سلام.csv')
df = df['0'].values.tolist()
for i in list1:
    if i in df:
        shutil.move('E://project//data2//validated//'+i,'E://project//data2//true class')
```

در این بخش از کد و با استفاده از فایل csv که پیش تر آن را ذخیره کردیم و حاوی اسامی فایل هایی که کلمه کلیدی در آنها گفته شده است بود، این فایل ها را به یک فولدر جداگانه انتقال میدهیم. برای اینکار از تابع move از

کتابخانه ی shutil استفاده میکنیم که با گرفتن آدرس مقصد به عنوان ورودی اول و آدرس مبدا به عنوان ورودی دوم، فایل در آدرس مبدا را به آدرس مقصد انتقال میدهد.

۳-۴ استخراج و جداسازی کلمات کلیدی

برای استخراج کلمات کلیدی و جداکردن آنها در یک فولدر جداگانه کد extract_keys.py پیاده سازی شده است که این عملیات به فرم تابع در این کد پیاده سازی شده است.

```
def extract_filenames(path,key):
    '''
    Parameters
    ----------
    path : str
        path of audio script file.
    key : str
        keyword that we want to be extracted.

    Returns
    -------
    saves a csv file containing path of audio files that contained the keyword.
```

تابع extract_filenames که همانند کد پیاده شده در قسمت قبل اسامی فایل هایی که حاوی کلمات کلیدی هستند را در یک فایل csv ذخیره میکرد عمل میکند. این تابع دو ورودی دارد که ورودی اول با نام path میباشد که آدرس فایل رونوشت های فایل های صوتی را میگیرد و ورودی دوم با نام key که کلمه کلیدی مد نظر برای استخراج را مشخص میکند.

```
df = pd.read_table(path)
ll = []
for i in range(len(df)):
    print(str(np.round( (i/len(df))*100,2) ) + ' % completed')
    l = df.iloc[i]['sentence'].split(' ')
    if key in l:
        ll.append(df.iloc[i]['path'])
ll = pd.DataFrame(ll)
ll.to_csv('E://project//data2//'+key+'.csv',index = False )
```

در این قسمت همانند قبل تمامی سطر های فایل رونوشت چک میشوند که آیا کلمه کلیدی در آن وجود دارد و اگر وجود داشت نام آن را ذخیره کند.

```
def move_filenames(keys_path,dest_path,files_path):
    '''
    Parameters
    ----------
    keys_path : str
        csv file containing keyword filenames.
    dest_path : str
        name of destination folder.
    files_path : str
        path of audio files.
    Returns
    -------
    moves all files that contain the keyword to a seperate folder.
```

تابع بعدی move_filenames میباشد که فایل اسامی را گرفته و فایل های صوتی داخل آنرا به فولدر مقصد انتقال میدهد. ورودی های این تابع به صورت زیر میباشد:

- keys_path: آدرس فایل حاوی رونوشت فایل های صوتی

- dest_path: آدرس فولدر مقصد که فایل ها به آن انتقال داده میشوند
- files_path: آدرس فولدر حاوی فایل های صوتی

```
df = pd.read_csv(keys_path)['0'].values.tolist()
list1 = os.listdir(files_path)
for i in range(len(list1)):
    print(str( np.round((i/len(list1))*100,2) ) + '% completed.')
    if list1[i] in df:
        shutil.move('E://project//data2//validated//'+list1[i],'E://project//data2//'+dest_path)
```

در این قسمت هم مانند قبل و با استفاده از تابع move از کتابخانه shutill فایل ها به فولدر مقصد انتقال میابند.

برای جداکردن کلمات کلیدی به صورت تک کلمه و در فایل هایی به طول ۱ ثانیه کد label.py پیاده سازی شد. تا اینجای کار فایل هایی که در جملات آنها کلمه کلیدی گفته شده است را در فولدری جداگانه قرار دادیم اما برای آموزش مدل باید فایل های صوتی ۱ ثانیه ای و حاوی تنها کلمه ی کلیدی را داشته باشیم. فایل های فعلی شامل جملات بازگو شده توسط افراد میباشد که در این جملات چندین کلمه گفته شده است و طول آنها بیش از ۱ ثانیه میباشد. برای جداسازی کلمات کلیدی، کدی پیاده سازی شده که همه ی این فایل ها را به تکه های کوچکتری تقسیم میکند و با پخش کردن آنها، لیبل و برچسب را از ورودی دریافت میکند. در ادامه به جزئیات این کد میپردازیم.

ابتدا کتابخانه ها و توابع مورد استفاده در کد را اضافه میکنیم.

```
import os
import librosa
import soundfile as sf
import pandas as pd
import numpy as np
from pydub import AudioSegment
import simpleaudio
import shutil
```

از کتابخانه ی librosa برای خواندن فایل های صوتی استفاده شده است. از تابع AudioSegment از کتابخانه pydub برای خواندن فایل صوتی استفاده شده است. از کتابخانه simpleaudio برای پخش فایل صوتی استفاده شده است.

```
def cut_audio(path,sr=22050):
    '''
    Parameters
    cuts audio files into 0.6 seconds intervals by hop length = 0.5 seconds
    and stores it in "cut" folder
    ----------
    path : str
        path of the folder containing audio files
    sr : int, optional
        sample rate. The default is 22050.

    Returns
    -------
    None.
    '''
```

تابع cut_audio برای تقسیم فایل های صوتی به قسمت های ۰,۶ ثانیه ای میاشد. دلیل اینکار بجای تقسیم فایل ها به تکه های ۱ ثانیه ای این بود که در قسمت های یک ثانیه ای معمولا بیش از یک کلمه گفته میشد و برای اینکه

تکه ها حتما حاوی تنها یک کلمه باشند، طول آنها را به ۰,۶ تغییر دادیم. هر یک از این قسمت ها هم با گام هایی به طول ۰,۵ ثانیه در طول سیگنال اصلی حرکت میکنند تا تکه ها با هم اشتراک داشته باشند و کلمه ای در وسط بازگو شدن آن قطع نشود و از دست نرود.

ورودی های این تابع به صورت زیر میباشند:

- path: آدرس فولدری که شامل فایل های صوتی میباشد که ما میخواهیم آنها را به تکه های ۰,۶ ثانیه ای تقسیم کنیم.
- sr: نرخ نمونه برداری که با استفاده از آن تعداد نمونه ها در ۰,۶ ثانیه یا ۰,۵ ثانیه محاسبه میشود.

خروجی این تابع، دخیره سازی تکه های بدست آمده از هر فایل در فولدری با نام cut در همان فولدر که آدرس آن به ورودی داده شد، میباشد.

```
li = os.listdir(path)
dir1 = os.getcwd()

for i in li:
    df,sr = librosa.load(path+'//'+i)
    c = 0
    for j in range(0,len(df),11025):    # hop by 5/10 of sample rate
        if j + 13230 < len(df):    # 0.6 seconds
            df1 = df[j:j+13230]
            sf.write(path+'//cut//'+i.split('.')[0]+'#'+str(c)+'.wav',df1,22050)
            c += 1
```

نحوه ی کار این تابع را در تصویر بالا مشاهده میکنید. ابتدا لیست فایل های موجود در فولدر در متغیر li ذخیره میشود. سپس حلقه ای بر روی این لیست تعریف میشود که ابتدا هر فایل توسط تابع load از کتابخانه ی librosa

خوانده میشود و این سیگنال به تکه هایی به طول ۰,۶ (۰,۶ * ۲۲۰۵۰ = ۱۳۲۳۰) و با گام های ۰,۵ ثانیه ای (۰,۵ * ۲۲۰۵۰ = ۱۱۰۲۵) تقسیم شده و این تکه ها با استفاده از تابع write از کتابخانه ی soundfile در فولدری با نام cut ذخیره میشود.

```
def label_sounds(path):
    '''
    plays each audio file and asks you for its label

    Parameters
    ----------
    path : str
        path of the folder containing our audio files

    Returns
    -------
    None.
```

تابع دیگر این کد label_sounds نام دارد که با گرفتن آدرس یک فولدر، تمامی فایل های صوتی داخل آن را به ترتیب پخش میکند و از ورودی یک عدد دریافت میکند که اگر این عدد برابر با ۱ بود، نام این فایل را بعنوان فایلی که کلمه کلیدی در آن گفته شده است ذخیره میکند.

```
li = os.listdir(path)
trues = []
for i in range(len(li)):
    print(np.round(i/len(li) * 100,2))
    audio = AudioSegment.from_wav(path+'//'+li[i])
    playback = simpleaudio.play_buffer(
            audio.raw_data,
            num_channels=audio.channels,
            bytes_per_sample=audio.sample_width,
            sample_rate=audio.frame_rate
            )

    p = input('please enter the label for the above audio:')
    if p == '1':
        playback.stop()
        trues.append(li[i])
        print(trues)
    else:
        playback.stop()
    print('------------------------------')
    print()

df = pd.DataFrame(trues)
df.to_csv('extracted_trues.csv',index=False)
```

نحوه کار این تابع را در تصویر بالا مشاهده میکنید. ابتدا لیست فایل های داخل فولدر خوانده میشود. سپس لیستی با نام trues ایجاد میشود که در نهایت اسامی فایل هایی که حاوی کلمه کلیدی میباشند را در خود نگه میدارد. در قسمت بعدی یک حلقه بر روی فایل ها زده میشود که در هر تکرار برای هر فایل، ابتدا فایل خوانده میشود و سپس توسط تابع play_buffer از

کتابخانه ی simpleaudio پخش میشود. در بخش بعدی ورودی از کاربر گرفته میشود. اگر این ورودی برابر با عدد ۱ بود، نام فایل فعلی به لیست trues اضافه میشود و حلقه ادامه پیدا میکند. در انتهای تابع لیست trues در قالب یک دیتافریم از کتابخانه ی pandas تبدیل میشود و سپس در فرمت csv ذخیره میشود.

۴-۴ ویژگی های قابل استخراج و استفاده در سیگنال های صوتی

در این قسمت به انواع ویژگی های قابل استخراج از سیگنال های صوتی که در مسائل یادگیری ماشین کاربرد دارند میپردازیم.

برای یادگیری الگوها با استفاده از مدل های یادگیری ماشین و یادگیری عمیق به داده هایی از جنس عددی نیاز داریم. سیگنال صوتی به صورت یک موج مکانیکی و پیوسته در زمان میباشد. در کامپیوتر ها برای پردازش صوت سیگنال پیوسته زمان آن را با استفاده از نمونه برداری به سیگنال گسسته تبدیل میکنند. پیش تر به نرخ نمونه برداری اشاره شد که این نرخ تعداد نمونه ها در یک ثانیه سیگنال را مشخص میکنند. در ادامه به ویژگی های قابل استخراج از سیگنال صوتی میپردازیم.

۴-۴-۱ سیگنال صوتی خام

فایل های صوتی در کامپیوتر ها به شکل دنباله ای از اعداد ذخیره میشوند که این اعداد نشان دهنده ی دامنه ی موج در هر نمونه میباشند. این ویژگی (دامنه موج) در حوزه ی زمانی میباشد و اطلاعاتی در رابطه با تغییرات دامنه در طول زمان به ما میدهد. از این سیگنال میتوان به عنوان ورودی مدل های یادگیری ماشین استفاده کرد اما این نوع سیگنال اطلاعاتی از جمله فرکانس صوت را در خود ندارند. برای استخراج این سیگنال در پایتون میتوان به صورت زیر عمل کرد:

```
signal,sr = librosa.load('filename.wav')
```

با استفاده از تابع load از کتابخانه ی librosa میتوان سیگنال دامنه/زمان را از فایل های صوتی استخراج کرد. این تابع آدرس فایل را بعنوان ورودی دریافت میکند و سیگنال و نرخ نمونه برداری را بعنوان خروجی بازمیگرداند.

برای استخراج اطلاعات بیشتر از سیگنال در حوزه ی زمان میتوان به صورت زیر عمل کرد:

فریم بندی سیگنال در زمان و سپس استخراج ویژگی از هر فریم که در بخش های بعدی به آن میپردازیم.

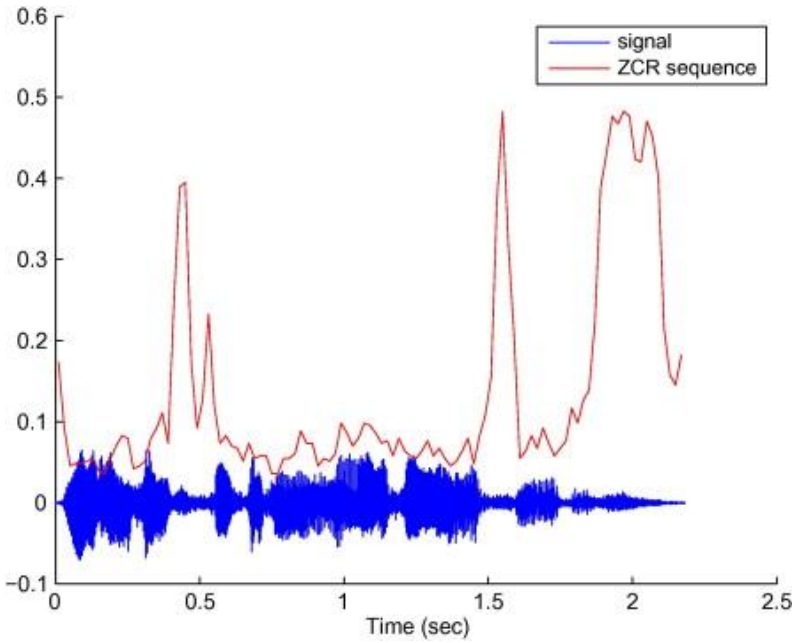
فریم بندی سیگنال:

در این پروسه سیگنال به قسمت هایی با طول مساوی تقسیم میشود. به هریک از این قسمت ها فریم گفته میشود. طول هر فریم معمولا بین ۲۵۶ تا ۸۱۹۲ میباشد.

zero-crossing rate ۲-۴-۴

برای هر فریم تعداد دفعاتی که سیگنال در آن فریم از محور افقی عبور میکند را محاسبه میکنیم و بعنوان ویژگی هر سیگنال استفاده میکنیم.

از این ویژگی در مسائلی که هدف آنها تشخیص بخش هایی از صوت که مکالمه در آن صورت گرفته مورد استفاده قرار میگیرد چراکه دامنه در صوتی که مکالمه ی انسان در آن وجود دارد بر خلاف صدای محیط، به دفعات زیاد بین مقادیر مثبت و منفی تغییر میکند.



۳-۴-۴ Root-mean-square energy

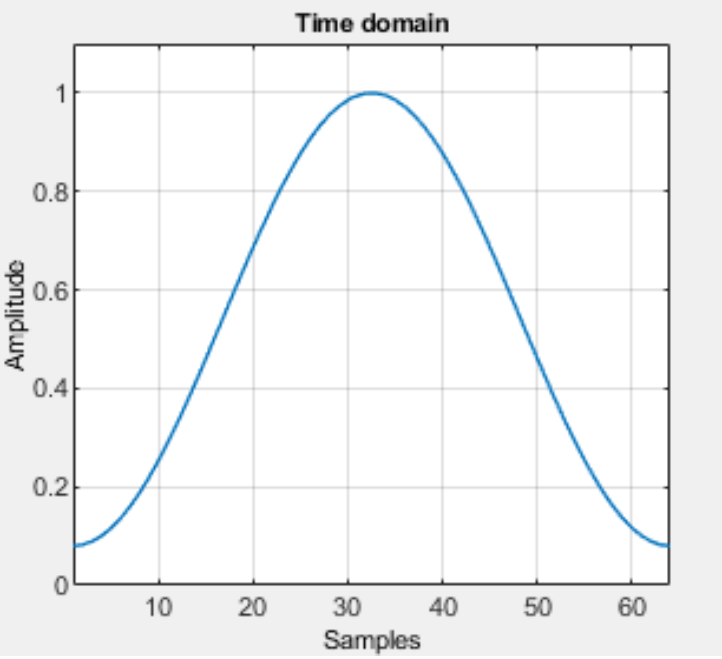
برای استخراج این ویژگی رادیکال مجموع مربعات مقادیر دامنه در هر فریم محاسبه میشود. مجموع مربعات دامنه، انرژی سیگنال را نشان میدهد. از موارد استفاده این ویژگی میتوان به مسائل تشخیص ژانر موسیقی اشاره کرد.

$$RMS = \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{N}}$$

از دیگر دسته ویژگی ها که قابل استخراج از سیگنال های صوتی میباشد میتوان به ویژگی ها در حوزه ی فرکانسی اشاره کرد. برای رفتن از یک سیگنال در حوزه زمان به حوزه ی فرکانس باید ابتدا همانند قبل سیگنال را به فریم ها تقسیم کرد. سپس با گرفتن تبدیل فوریه از سیگنال، میتوان به حوزه فرکانس رفت. اما این روش مشکلاتی دارد. اولین مشکل که spectral leakage نام دارد، ناپیوستگی و بریدگی در ابتدا و انتهای هر فریم باعث اضافه شدن مولفه هایی با فرکانس بالا در نتیجه میشود که در اصل در سیگنال اصلی نبوده و به دلیل همین مشکل میباشد. برای حل این مشکل میتوان از توابع windowing استفاده کرد که باعث کم شدن اثر مقادیر انتها و ابتدای فریم میشود و یک سیگنال متناوب را به ما میدهد.

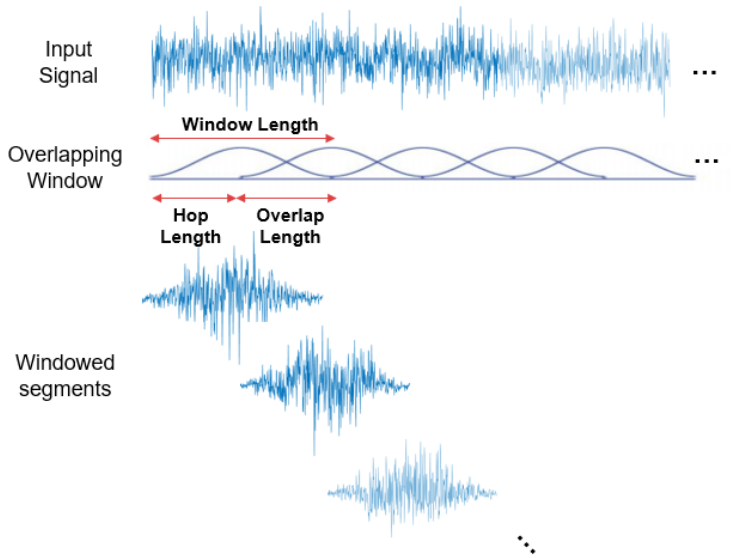
یکی از توابع متداول برای اینکار تابع Hann window نام دارد که فرمول آن به صورت زیر میباشد:

$$w(k) = 0.5 - 0.5\cos\left(\frac{2\pi k}{N-1}\right)$$



سپس با ضرب این تابع در هر فریم از سیگنال اصلی، مشکل spectral leakage را بر طرف میکنیم. اما با این روش یک مشکل جدید به وجود می آید. بدلیل اینکه اطلاعات در ابتدا و انتهای هر فریم از بین میرود، ما بخش زیادی از اطلاعات سیگنال خود را از دست میدهیم. برای حل این مشکل میتوان از فریم هایی استفاده کرد که با هم اشتراک دارند؛ به این صورت که به جای اینکه هر فریم دقیقا از آخرین نمونه فریم قبلی شروع شود، میتواند مقداری با فریم قبلی اشتراک داشته باشد. برای رسیدن به این مهم میتوان گام های جلو رفتن فریم ها را کوچکتر از طول هر فریم در نظر بگیریم. به

طول این گام ها hop length گفته میشود و با استفاده از این روش فریم ها با هم اشتراک دارند و دیگر اطلاعاتی از دست نمیرود. در تصویر زیر میتوان این فرایند را مشاهده کرد.



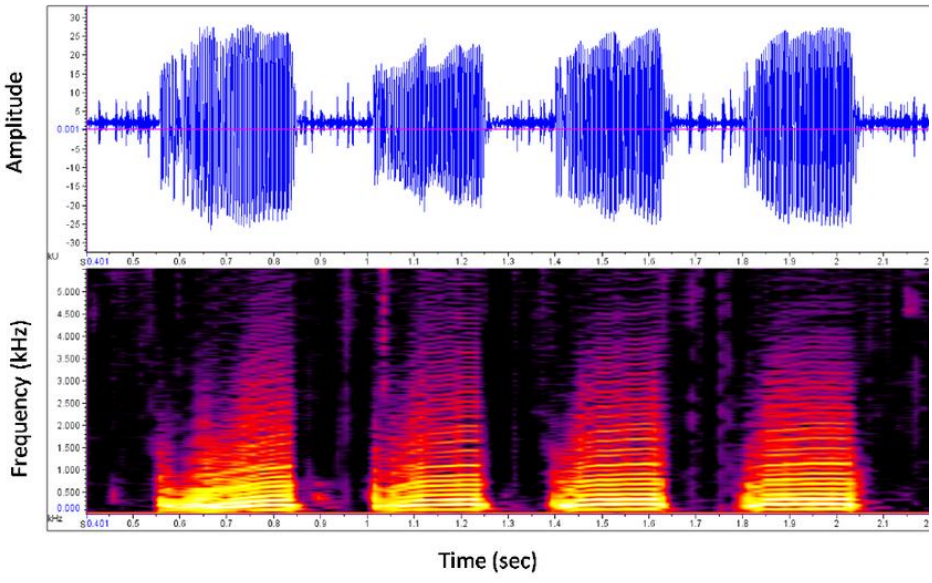
پس از فریم بندی سیگنال با گام های کوچکتر از طول فریم و اعمال تابع windowing روی فریم ها، میتوان تبدیل فوریه را محاسبه کرد و سیگنال را به حوزه فرکانس برد. بدلیل اینکه سیگنال ما گسسته میباشد میتوانیم از تبدیل فوریه گسسته زمان استفاده کنیم.

Spectrogram ۴-۴-۴

اسپکتروگرام (spectrogram) یک نمایشی از طیف فرکانسی سیگنال در طول زمان است. اسپکتروگرام یک نقشه رنگی هست که برحسب زمان و

فرکانس نمایش داده می شود و به ما کمک می‌کند تا متوجه شویم که طیف فرکانسی سیگنال در طول زمان به چه صورت تغییر می‌کند.

اسپکتروگرام از روی خروجی تبدیل فوریه زمان کوتاه (Short Time Fourier Transform) بدست می آید. برای بدست آوردن آن بر روی هر فریم از سیگنال تبدیل فوریه زمان کوتاه اعمال میشود و به ازای هر فرکانس موجود در هر فریم، یک عدد که ضریب آن فرکانس در آن فریم را نشان میدهد را بازمیگرداند. پس برای هر فریم این ضرایب بدست آمده را داریم که اگر تمامی این خروجی ها را به همان ترتیب فریم ها کنار هم بچینیم، یک ماتریس از ضرایب بدست می آید که میتوان به آن مانند یک تصویر دو بعدی نگاه کرد که هر پیکسل مقداری را در خود نگهداری میکند. از این ویژگی میتوان در شبکه های عصبی کانوولوشنال استفاده کرد چرا که همانند ورودی آن شبکه، این ویژگی را نیز میتوان مانند یک تصویر به شبکه داد. در تصویر زیر میتوان یک نمونه از خروجی را مشاهده کرد:

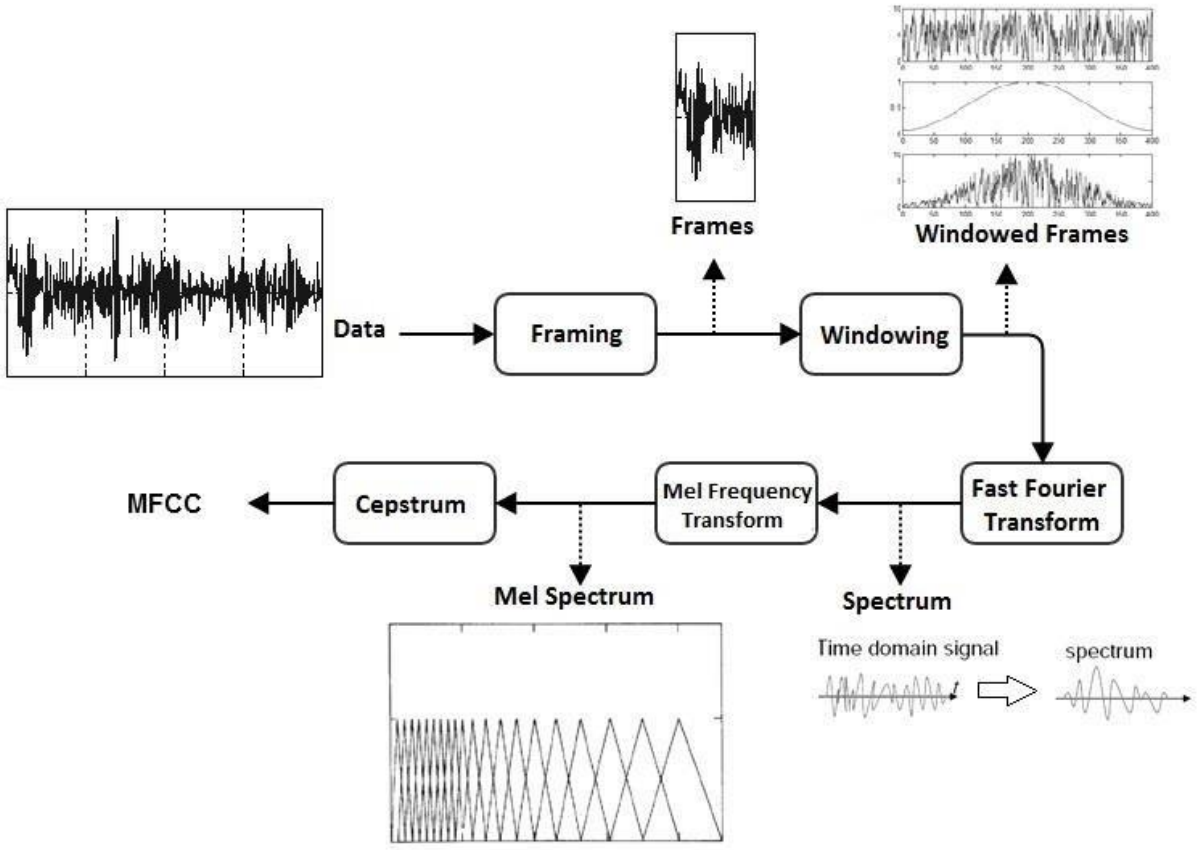


۵-۴-۴ Mel-Frequency Cepstral Coefficients

متداول ترین و کارا ترین ویژگی ها برای بازشناسی گفتار، Mel-Frequency Cepstral Coefficients یا به اختصار MFCC ها هستند. ایده اصلی این ویژگی از نحوه ی دریافت و درک صوت توسط انسان گرفته شده است. Mfcc طیف توان یک صوت را با استفاده از تبدیل کسینوسی خطی لگاریتم طیف توان در مقیاس مل نشان میدهد. فرمول مقیاس مل به صورت زیر میباشد:

$$M(f) = 1125 \ln(1 + \frac{f}{700})$$

که در آن f مقدار فرکانس و M مقدار مل متناظر با آن را نشان میدهد. ضرایب MFCC را میتوان تبدیل کسینوسی لگاریتم انرژی حاصل از اعمال فیلتر بانک مل بر طیف سیگنال که windowing روی آن اعمال شده، تعریف کرد. معمولا ۹ تا ۱۳ ضریب اول از این تبدیل بیشتر اطلاعات سیگنال صوتی را در خود نگهداری میکنند و در مسائل یادگیری ماشین معمولا از ۱۳ سیگنال اول استفاده میشود. مراحل محاسبه این ضرایب به صورت زیر میباشد:



در تصویر بالا تمامی مراحل تبدیل سیگنال خام در حوزه زمان تا تبدیل آن به MFCC آورده شده است. مرحله اول تبدیل سیگنال به فریم ها میباشد که این مرحله با نام framing در تصویر آورده شده است. مرحله بعدی اعمال تابع windowing روی این فریم هاست که در تصویر با نام windowing آورده شده است. سپس از هر فریم یک تبدیل فوریه گرفته میشود که با این تبدیل spectrogram بدست می آید. سپس فرکانس های موجود در spectrogram را به مقیاس مل میبریم و آنرا از فیلتر بانک مل عبور میدهیم تا cepstrum بدست بیاید. سپس از این ضرایب تبدیل فوریه معکوس میگیریم و ضرایب mfcc در نهایت حاصل میشود. معمولا ۱۳ ضریب

اول بدست آمده از هر فریم را برای استفاده در مسائل یادگیری ماشین جدا میکنند.

۴-۵ ایجاد دیتاست

در این بخش به توضیح کد make_dataset.py که برای ایجاد مجموعه دادگان قابل استفاده توسط مدل ها میباشد، میپردازیم.

ابتدا کتابخانه های استفاده شده در این کد را اضافه میکنیم.

```
import librosa
import os
import json
import numpy as np
import random
```

کتابخانه ی librosa برای خواندن فایل های صوتی، کتابخانه ی json برای ذخیره سازی دیتاست نهایی در فرمت json و کتابخانه ی random.

```
def chop_up(data):
    chops = []
    for j in range(0,len(data),11025):    # hop by 0.5 second
        if j + 22050 < len(data):    # 1 second
            data1 = data[j:j+22050]
            chops.append(data1)
    return chops
```

یکی از توابع که در این کد پیاده سازی شده است تابع chop_up میباشد که جلوتر و در ایجاد دیتاست از آن استفاده شده است. کاربرد این تابع تقسیم یک سیگنال صوتی به قسمت هایی به طول ۱ ثانیه و با گام هایی به طول 0,5 ثانیه میباشد. دلیل استفاده از این تابع این است که فایل های کلاس false (فایل هایی که کلمه کلیدی در آنها گفته نشده است) طولی بیشتر از ۱ ثانیه دارند و چون ورودی مدل های کانوولوشنال باید ابعاد یکسانی داشته باشند، باید به قطعات ۱ ثانیه ای تبدیل شوند. این تابع سیگنال اصلی را به عنوان ورودی دریافت میکند و آنرا به قسمت های یک ثانیه ای تبدیل میکند و این قسمت ها را در لیستی با نام chops بازمیگرداند.

```
def create_dataset(path):

    data = {
            "map": ['false','true'],  # what keyword each number represent
            "labels": [],     # label of each audio file
            "MFCC": [],       # MFCC of each audio file
            "file": []        # path of each audio file
            }
```

تابع اصلی که برای ایجاد دیتاست مورد استفاده قرار میگیرد create_dataset نام دارد. این تابع یک ورودی دارد که آدرس فولدری که شامل فولدر کلاس های نهایی میباشد را نشان میدهد. در این فولدر به تعداد کلاس های نهایی فولدر وجود دارد که هم نام کلاس ها میباشند. در این فولدر ها فایل های مربوط به هر کلاس آورده شده است. در مسئله ما این کلاس ها true (داده هایی که کلمه کلیدی در آن گفته شده) و false (داده هایی که

کلمه کلیدی در آن نگفته شده است) نام دارند. در ابتدای تابع یک دیکشنری با نام data ایجاد میشود که در نهایت بعنوان دیتاست نهایی ذخیره میشود. کلید های این دیکشنری به صورت زیر میباشند:

- map: مقادیر این کلید لیستی از کلاس های موجود در دیتاست را نشان میدهد.
- labels: لیستی از برچسب های هر داده موجود در دیتاست.
- MFCC: لیستی از ویژگی های استخراج شده برای هر داده.
- File: لیستی از نام فایل های داده های موجود در دیتاست.

تمامی این لیست ها به یک ترتیب میباشند. به این معنا که اولین عضو از لیست file نام فایل اولین داده در دیتاست را نشان میدهد که برچسب این داده در اولین عضو لیست labels آورده شده است و ویژگی های استخراج شده از این فایل در اولین عضو لیست MFCC آمده است.

```
for i in range(len(data['map']) ):
    files = os.listdir(path+'//' + data['map'][i])
    random.shuffle(files)
    for f in files:
        filename = path+'//' + data['map'][i] + '//' + f
        print(filename)
```

ابتدا بر روی فولدر ها یک حلقه ایجاد میکنیم. نام فولدر ها همانطور که در قسمت قبل گفته شد در لیست map آورده شده است. در این حلقه ابتدا لیست فایل های داخل این فولدر در متغیر files ریخته میشود و ترتیب این

لیست توسط تابع shuffle از کتابخانه ی random به یک ترتیب تصادفی تبدیل میشود چراکه در ابتدا بر اساس الفبا مرتب شده اند و فایل هایی با گوینده یکسان پشت سر هم قرار گرفته اند. سپس یک حلقه روی اسامی فایل ها زده میشود تا ویژگی ها و دیگر اطلاعات از آنها استخراج شوند.

```
signal , sr = librosa.load(filename)
if data['map'][i] == 'false' and len(signal) >= 22050:
    chps = chop_up(signal)
    count += len(chps)
    if count >= 6000:
        break
    for c in range(len(chps)):
        mfcc = librosa.feature.mfcc(chps[c], sr=22050, n_mfcc=13, hop_length=512, n_fft=2048)
        data['labels'].append(i)
        data['MFCC'].append(mfcc.T.tolist())
        data['file'].append(filename)
elif len(signal)==22050:
    mfcc = librosa.feature.mfcc(signal, sr=22050, n_mfcc=13, hop_length=512, n_fft=2048)
    data['labels'].append(i)
    data['MFCC'].append(mfcc.T.tolist())
    data['file'].append(filename)
```

در این قسمت سیگنال صوتی از فایل توسط تابع load از کتابخانه librosa خوانده میشود. سپس چک میکنیم که آیا کلاس این فایل false هست یا نه. اگر کلاس آن false بود به این معناست که طول فایل صوتی آن از ۱ ثانیه بیشتر است و نیاز است که آنرا به قسمت های ۱ ثانیه ای تقسیم کنیم. از ابتدای حلقه ی اول یک متغیر با نام count که مقدار اولیه آن ۰ میباشد تعریف کردیم تا نسبت کلاس ها را با هم توسط این متغیر کنترل کنیم. برای مثال به ازای هر داده با کلاس false که اضافه میکنیم مقدار این متغیر یکی زیاد میشود. بدلیل اینکه تعداد داده ها با کلاس true بسیار کمتر از کلاس false میباشد، نیاز است که نسبت تعداد داده های هر کلاس در دیتاست

کنترل شود. اگر مقدار متغیر count برای مثال از ۶۰۰۰ بیشتر شد، دیگر به داده های این کلاس اضافه نمیکنیم. سپس برای هر قسمت یک ثانیه ای، ضرایب mfcc را استخراج میکنیم. برای اینکار از تابع mfcc از کتابخانه ی librosa استفاده میکنیم. ورودی های این تابع به صورت زیر میباشد:

- ورودی اول که سیگنالی که میخواهیم این ضرایب را از آن استخراج کنیم را نشان میدهد.
- Sr که نرخ نمونه برداری سیگنال را نشان میدهد.
- n_mfcc که تعداد ضرایبی که میخواهیم را مشخص میکند.
- hop_length: طول گام یا همان hop length را مشخص میکند.
- n_fft: طول هر فریم را مشخص میکند.
- تمامی این ورودی ها و نحوه ی استخراج این ویژگی در ۴-۴ توضیح داده شد.

بدلیل اینکه مقدار hop_length برابر با ۵۱۲ قرار گرفته، خروجی این تابع یک ماتریس با ابعاد ۴۴ (22050 / 512) در ۱۳ (تعداد ضرایب) میباشد. پس هر سیگنال صوتی به یک ماتریس 44x33 تبدیل میشود که این ویژگی به لیست MFCC از دیکشنری دیتا اضافه میشود. باقی اطلاعات این فایل نیز به دیکشنری اضافه میشود و سپس حلقه برای باقی فایل ها اجرا میشود.

```
with open(path+'//data.json','w') as f:
    json.dump(data,f,indent=4)
```

در انتها دیکشنری data به فرمت json تبدیل شده و در فایلی با نام data.json ذخیره میشود.

همانطور که گفته شد، تعداد فایل ها با کلاس true (فایل هایی که در آن کلمه کلیدی گفته شده اند) بسیار کمتر از کلاس دیگر میباشد. علاوه بر محدود کردن تعداد فایل های کلاس false که پیشتر توضیح داده شد، اقدامات دیگری برای کنترل این نامتوازنی در تعداد داده های هر کلاس انجام گرفت که در بخش بعدی به آنها میپردازیم.

۴-۶ راه حل های موجود برای دیتاست نامتوازن

مشکل نامتوازن بودن کلاس های داده ها در دیتاست برای مسائل دسته بندی میتواند در عملکرد مدل ها تاثیر منفی بگذارد. برای مثال اگر کلاس true ۰,۱ داده ها را تشکیل بدهد، مدل میتواند همیشه کلاس false را پیشبینی کند و دقت آن هم ۰,۹ بشود در حالی که برای همه ی اعضای کلاس true اشتباه پیشبینی کرده است. برای حل این مشکل راه های مختلفی وجود دارد که در این قسمت به آن ها میپردازیم. یکی از راه حل های این مشکل اضافه کردن دیتا برای کلاس با تعداد کمتر میباشد که از این راه نیز در این پروژه استفاده شده است (در قسمت ۴-۱-۱ توضیح داده شد).

۴-۶-۱ Upsampling

به فرایند اضافه کردن تعداد نمونه ها در کلاسی که تعداد کمتری دارد گفته میشود. راه های زیادی برای اینکار وجود دارد که یکی از این ها تکرار داده ها در کلاس کمتر میباشد. به این معنا که هر نمونه عضو این کلاس را تا جایی که نسبت کلاس ها درست شود، چندین بار کپی میکنیم و به دیتاست اضافه میکنیم. این روش میتواند به مشکل overfitting منجر شود. روش دیگر برای upsampling تولید داده از روی داده های فعلی میباشد. برای اینکار میتوانیم با اعمال تغییرات روی داده های فعلی، داده های جدیدی را تولید کنیم و تعداد داده ها در این کلاس را افزایش دهیم. برای مثال در داده های تصویری میتوان با چرخش یا حذف بخشی از تصویر یا اضافه کردن نویز به تصویر به این مهم دست یافت. در داده های صوتی نیز میتوان با تغییر pitch، اضافه کردن نویز، تغییر سرعت داده های جدیدی تولید کرد. در ادامه کدی که برای پیاده سازی این روش نوشته شد را بررسی میکنیم.

برای استفاده از upsampling بر روی داده های موجود در دیتاست، کد upsample.py پیاده سازی شده است.

```
import librosa as lib
import numpy as np
import os
import soundfile as sf
import shutil
```

ابتدا کتابخانه های مورد استفاده در این کد را اضافه میکنیم. تمامی این کتابخانه ها در بخش های قبل توضیح داده شدند.

```
def noise(sig,noise_factor):
    '''
    Parameters
    ----------
    sig : numpy array of audio signal
    noise_factor : float [0,1]
        significance of noise.

    Returns
    -------
    augmented_data : numpy array of augmented audio signal.
        returns the audio signal with random nois added to it.

    '''
    noise = np.random.randn(len(sig))
    augmented_data = sig + noise_factor * noise
    augmented_data = augmented_data.astype(type(sig[0]))
    return augmented_data
```

تابع noise که برای اضافه کردن نویز به سیگنال صوتی پیاده سازی شده است. این تابع دو ورودی دارد که ورودی اول آن سیگنال صوتی و ورودی دوم آن یک فاکتور نویز میباشد که تاثیر نویز روی سیگنال اصلی را کنترل میکند. برای اضافه کردن نویز یک لیست از اعداد رندوم بین -۱ تا ۱ به طول سیگنال اصلی با استفاده از تابع randn از کتابخانه numpy انتخاب میکنیم و آن را به سیگنال اضافه میکنیم. در انتها این سیگنال تغییر یافته بعنوان خروجی تابع بازگردانده میشود.

```
def pitch(sig,sr,pitch_factor):
    '''
    Parameters
    ----------
    sig : numpy array of audio signal

    sr : int
        sample rate
    pitch_factor : float [0,1]

    Returns
    -------
    numpy array of pitch shifted audio signal


    '''
    return lib.effects.pitch_shift(sig, sr, pitch_factor)
```

تابع دیگر پیاده شده در این کد تابع pitch میباشد که برای تغییر pitch سیگنال صوتی مورد استفاده قرار میگیرد. برای اینکار از تابع pitch_shift از کتابخانه ی librosa استفاده شده است.

```
def extend(path,sample_rate):
    '''
    extends audio files by adding silence to reach 1 second of duration based on sample rate

    Parameters
    ----------
    path : str
        path of the folder containing audio files
    sample_rate : int
        sample rate of audio files

    Returns
    -------
    None.

    '''
```

در این تابع سیگنال هایی با طول کمتر از ۱ ثانیه، با اضافه کردن دامنه صفر به ابتدا و انتهای آنها به صورت رندوم، به سیگنال های ۱ ثانیه ای تبدیل میشوند.

ورودی اول این تابع آدرس فولدری است که فایل های صوتی در آن قرار دارند. ورودی دوم نرخ نمونه برداری این فایل ها را مشخص میکند. هرف پیاده سازی این تابع تصحیح طول سیگنال هایی بود که در قسمت های قبلی با طول ۰,۶ ثانیه تقسیم شدند.

```
li = os.listdir(path)
for i in li:
    if len(i.split('.')) > 1 and i.split('.')[1] == 'wav':
        data,sr = lib.load(path+'\\'+i)
        data = list(data)
        short = sample_rate - len(data)
        left = np.random.randint(0,short)
        left = np.zeros(left)
        left = list(left)
        right = np.zeros(short-len(left))
        right = list(right)
        final = []
        final.append(left)
        final.append(data)
        final.append(right)
        f2 = []
        for j in final:
            for k in j:
                f2.append(k)
        f2 = np.array(f2)
        sf.write(path+'\\extended\\'+i.split('.')[0]+'_ext.wav',f2,sample_rate)
```

ابتدا اسامی فایل های این فولدر در لیستی ذخیره میشود تا بتوانیم روی این لیست یک حلقه بزنیم. در این حلقه، ابتدا سیگنال صوتی از روی فایل خوانده میشود و سپس با تولید یک عدد تصادفی، تعداد نمونه هایی که باید سمت چپ سیگنال (ابتدای سیگنال) مشخص میشود و به همین تعداد نمونه هایی با دامنه ۰ به سمت چپ سیگنال اضافه میکنیم. سپس نمونه های باقیمانده برای رسیدن به ۱ ثانیه را به سمت راست سیگنال (انتهای سیگنال) اضافه میکنیم و در انتها این سیگنال را با استفاده از تابع write از کتابخانه ی soundfile با فرمت .wav در فولدری با نام extended ذخیره میکنیم.

```
def upsample(path=None):
    '''
    up-samples the dataset by augmenting new data from existing audio files in the path.

    Parameters
    ----------
    path : str
        path of audio files.

    Returns
    -------
    up-samples the dataset and saves new files in "augmented" folder.
```

تابع upsample برای افزایش تعداد داده های کلاس کمتر به روش upsampling پیاده سازی شده است. ورودی این تابع آدرس فولدری است که شامل فایل های این کلاس بوده و خروجی آن ذخیره فایل های تولید شده در فولدری با نام augmented میباشد.

```
list1 = os.listdir(path)

for x in list1:
    if 'wav' not in x.split('.'):
        continue

    sig,sr = lib.load(path+'\\'+x)
    noisy = []
    pitchy = []
    for j in range(2):
        fac = np.random.uniform(low=0.0, high=0.0009)
        n_steps = np.random.uniform(-2,2)
        noisy.append(noise(sig,fac))
        pitchy.append(pitch(sig,sr,n_steps))

    # save augmented audio files
    name = x.split('.')[0]
    for i in range(len(noisy)):
        for k in range(2):
            sf.write(path+'//augmented//4//'+name+'_noise_'+str(i)+'_'+str(k)+'.wav',noisy[i],22050)
            sf.write(path+'//augmented//4//'+name+'_pitch_'+str(i)+'_'+str(k)+'.wav',pitchy[i],22050)
            sf.write(path+'//augmented//4//'+name+'_replicated_'+str(k)+'.wav',sig,22050)
```

ابتدا اسامی فایل های داخل فولدر در لیستی ذخیره میشود. سپس بر روی این لیست یک حلقه تعریف میشود. در این حلقه ابتدا سیگنال اصلی از فایل

خوانده میشود. سپس برای این سیگنال به تعداد ۲ عدد سیگنال جدید با توابع noise و pitch تولید میشود. در انتها این سیگنال ها به همراه دو کپی از سیگنال اصلی در فولدر augmented ذخیره میشوند.

۲-۶-۴ downsampling

روش دیگر برای حل مشکل نامتوازن بودن دیتاست، downsampling نام دارد. در این روش تعداد نمونه ها در کلاسی که تعداد بیشتری دارد را با حذف نمونه ها از آن کم میکنیم. این روش در پروژه ما قابل استفاده نبود چراکه شبکه های عصبی برای یادگیری نیاز به مقادیر زیادی داده دارند و این روش تعداد داده ها را کم میکند و برای این دست مسائل قابل استفاده نمیباشد.

فصل پنجم

۵-۱ ارائه معماری برای مدل ها

در این قسمت به جزئیات و معماری مدل ها میپردازیم. برای شروع کار از یک معماری معمول و متداول برای شبکه ها استفاده کردیم که جلوتر به جزئیات آن اشاره میشود. سپس به بررسی عملکرد مدل ها میپردازیم و در نهایت راه حل های استفاده شده برای بهبود عملکرد مدل را بررسی میکنیم.

۵-۱-۱ مدل CNN

در این قسمت به جزئیات مدل شبکه کانوولوشنال میپردازیم. اولین معماری که برای این مدل پیاده سازی شد به صورت زیر بود:

Layer (type)	Output Shape	Param #
conv2d_30 (Conv2D)	(None, 42, 11, 64)	640
batch_normalization_30 (Batc	(None, 42, 11, 64)	256
max_pooling2d_30 (MaxPooling	(None, 21, 6, 64)	0
conv2d_31 (Conv2D)	(None, 19, 4, 32)	18464
batch_normalization_31 (Batc	(None, 19, 4, 32)	128
max_pooling2d_31 (MaxPooling	(None, 10, 2, 32)	0
conv2d_32 (Conv2D)	(None, 9, 1, 32)	4128
batch_normalization_32 (Batc	(None, 9, 1, 32)	128
max_pooling2d_32 (MaxPooling	(None, 5, 1, 32)	0
flatten_10 (Flatten)	(None, 160)	0
dense_20 (Dense)	(None, 64)	10304
dropout_10 (Dropout)	(None, 64)	0
dense_21 (Dense)	(None, 10)	650

Total params: 34,698
Trainable params: 34,442
Non-trainable params: 256

این معماری از ۱۳ لایه تشکیل شده است که این لایه ها به شرح زیر میباشند:

- لایه ورودی که یک لایه کانوولوشنال با ۶۴ فیلتر ۳ در ۳ میباشد.
- لایه بعدی که یک لایه batch normalization است. این لایه برای نرمال کردن خروجی های لایه قبلی میباشد.
- لایه max_pooling با فیلتر ۳ در ۳ و گام های (2x2)
- سه لایه ی بعدی تکرار لایه های بالا با این تفاوت که در لایه کانوولوشنال دارای ۳۲ فیلتر میباشد.
- سه لایه بعدی هم مانند سه لایه قبلی هستند. با این تفاوت که فیلتر های max_pooling آن ۲ در ۲ میباشد.
- لایه بعدی لایه ی flatten میباشد که ماتریس را به یک آرایه تک بعدی تبدیل میکند.
- لایه بعدی یک لایه ی fully connected با ۶۴ نرون و تابع فعالسازی relu میباشد.
- لایه بعدی یک لایه ی dropout با ۶۴ نرون و ضریب ۰,۳ میباشد. کاربرد این لایه غیر فعالسازی نرون ها با احتمال ۰,۳ در هر تکرار میباشد. اینکار برای جلوگیری از overfitting استفاده میشود.
- لایه نهایی هم یک لایه fully connected با ۲ نرون میباشد که همین لایه ی خروجی هست (۲ کلاس خروجی داریم). تابع فعالسازی این لایه softmax میباشد چراکه به یک احتمال برای هر کلاس نیاز داریم.

قبل از پرداختن به نحوه ی پیاده سازی این مدل باید به این نکته اشاره شود که این مدل با دقت ۹۹,۹۹ به درستی داده های آموزش و validation را کلاس بندی میکرد و این به دلیل overfit شدن مدل روی داده ها بود و مدل روی داده های تازه و ضبط شده به صورت دستی عملکرد خوبی نداشت.

به همین دلیل از یک معماری ساده تر استفاده شد که در بخش بعدی به جزئیات پیاده سازی آن میپردازیم.

۵-۲ پیاده سازی مدل

همانطور که در قسمت قبلی گفته شد، بدلیل مشکلات مدل قبلی، مدلی با معماری جدید پیاده سازی شد که در این قسمت به آن میپردازیم.

کد پیاده سازی مدل ها model.py نام دارد. ابتدا کتابخانه های مورد استفاده را به آن اضافه میکنیم.

```
import json
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow.keras as keras
import pandas as pd
```

تابع train_test_split از کتابخانه ی sklearn برای تقسیم دیتاست به دو بخش مجموعه دادگان آموزش و تست استفاده شده است. باقی کتابخانه ها در قسمت های قبلی توضیح داده شدند.

```
def prepare_data(path):

    with open(path) as f:
        data = json.load(f)
    X = data['MFCC']
    y = data['labels']
    X = np.array(X)
    y = np.array(y)
    X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=0.1)
    X_train, X_val, Y_train,Y_val = train_test_split(X_train,Y_train,test_size=0.1)
    return (X_train,X_test,Y_train,Y_test,X_val,Y_val)
```

در ابتدای کار تابع prepare_data برای آماده سازی دیتاست برای استفاده از آنها توسط مدل، پیاده سازی شده است. ورودی این تابع آدرس فایل دیتاست میباشد. ابتدا محتوای دیتاست (که در فرمت json میباشد که جزئیات آن در قسمت ۴-۵ توضیح داده شد) را از فایل آن میخوانیم و در متغیری با نام data ذخیره میکنیم. مدل ها به دو دسته دیتا نیاز دارند: ورودی ها که ویژگی های استخراج شده از سیگنال های صوتی هستند (کلید MFCC در دیتاست) و خروجی ها یا کلاس هر داده (کلید labels در دیتاست). این دو ابتدا در دو متغیر x و y ذخیره میشوند. سپس باید این دیتاست را به چند دسته تقسیم کرد: مجموعه دادگان آموزش، مجموعه دادگان تست و مجموعه validate. برای اینکار از تابع train_test_split استفاده شده است. ورودی این تابع با عنوان test_size برابر با ۰,۱ قرار داده شده به این معنا که ۰,۱ از کل داده ها بعنوان مجموعه تست و باقی داده ها به عنوان مجموعه آموزش (چراکه برای آموزش به داده های زیادی نیاز دارد) در نظر گرفته شده است. این ۰,۹ باقی نیز دوباره به دو قسمت تبدیل شده است

که اینبار ۰,۱ از این ۰,۹ بعنوان مجموعه validate در نظر گرفته شده است. سپس تمامی این مجموعه ها که در فرمت آرایه کتابخانه نامپای هستند، بعنوان خروجی تابع بازگردانده میشوند.

```
#-------------- loading the data -----------------------
(X_train,X_test,Y_train,Y_test,X_val,Y_val) = prepare_data(path+'\\data.json')
# adding number of channels to our data ( so far the data is: [# of segments,# coefs])
X_train = X_train[...,np.newaxis]
X_test = X_test[...,np.newaxis]
X_val = X_val[...,np.newaxis]
```

در بخش بعدی کد، با استفاده از تابع prepare_data دیتاست به مجموعه دادگان تقسیم میشود. به تمامی این داده ها یک بُعد اضافه شده است چرا که ورودی مدل cnn از سه بُعد تشکیل میشود که این ابعاد به شرح زیر میباشند:

[# segments, #coefficients, #channels]

داده های فعلی ما تنها حاوی دو بعد اول میباشند که تعداد قسمت های هر سیگنال و تعداد ضرایب استخراج شده را نشان میدهد. بعد آخر تعداد کانال های رنگی را نشان میدهد که در مسائل پردازش تصویر کاربرد دارد (برای تصاویر رنگی ۳ کانال قرمز، سبز و آبی وجود دارد و برای تصاویر سیاه سفید یک کانال) اما در مسئله ما تنها به یک کانال نیاز داشتیم چراکه در هر درایه از ماتریس تنها یک عدد ضریب وجود دارد. برای اضافه کردن این بعد از تابع newaxis از کتابخانه نامپای استفاده شده است.

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 42, 11, 32)        320
_________________________________________________________________
batch_normalization (BatchNo (None, 42, 11, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 21, 6, 32)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 20, 5, 16)         2064
_________________________________________________________________
batch_normalization_1 (Batch (None, 20, 5, 16)         64
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 3, 16)         0
_________________________________________________________________
flatten (Flatten)            (None, 480)               0
_________________________________________________________________
dense (Dense)                (None, 32)                15392
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 66
=================================================================
Total params: 18,034
Trainable params: 17,938
Non-trainable params: 96
```

مدل پیاده سازی شده در این کد یک نمونه ساده تر از مدل توضیح داده شده در بخش قبلی میباشد. دلیل این ساده سازی مشکل overfitting بود که در بخش بعدی به آن خواهیم پرداخت. برای مثال تعداد لایه های کانوولوشن یکی کمتر از مدل قبلی میباشد و تعداد فیلتر های آن هم کمتر از مدل قبلی است. در ادامه به پیاده سازی این مدل میپردازیم.

```
#********************************************* build the model ******
input_shape = (X_train.shape[1], X_train.shape[2], X_train.shape[3])
learning_rate = 0.0001

model = keras.Sequential()
```

ابتدا ابعاد ورودی مدل را از دیتاست ها بدست آورده در متغیر input_shape ذخیره میکنیم. سپس فاکتور نرخ یادگیری را برابر با عدد ۰,۰۰۰۱ قرار میدهیم. سپس مدل را توسط تابع Sequential از کتابخانه ی

keras ایجاد میکنیم. این مدل به صورتی میباشد که میتوان به آن با استفاده از توابع موجود در کتابخانه ی keras، لایه اضافه کرد.

```
#conv layer 1
model.add(keras.layers.Conv2D(32, (3,3), activation='relu',
          input_shape=input_shape,kernel_regularizer=keras.regularizers.l2(0.005)))
```

با تابع add از این کتابخانه امکان اضافه کردن لایه به شبکه وجود دارد. ابتدا یک لایه کانوولوشن به عنوان لایه ورودی به مدل اضافه میکنیم. این لایه شامل ۳۲ فیلتر با ابعاد ۳ در ۳ و تابع فعالسازی relu و نرم L2 با ضریب ۰,۰۰۵ (این L2_requlatizer باعث اضافه کردن یک پنالتی برای اندازه وزن ها به تابع خطا میشود که از بزرگ شدن وزن ها از حدی جلوگیری میکند) میشود.

```
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.MaxPool2D( (2,2), strides=(2,2) , padding='same' ))
```

سپس دو لایه batch_normalization و لایه max_pooling با فیلتر ۲ در ۲ به شبکه اضافه شده است.

```
#conv layer 2
model.add(keras.layers.Conv2D(16, (2,2), activation='relu',
          kernel_regularizer=keras.regularizers.l2(0.005)))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.MaxPool2D( (2,2), strides=(2,2) , padding='same' ))
```

در ادامه سه لایه مشابه با لایه های قبلی به شبکه اضافه میشود با این تفاوت که لایه کانوولوشن آن شامل ۳۲ فیلتر ۲ در ۲ میباشد.

```
# dense layer
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(32, activation='relu'))
model.add(keras.layers.Dropout(0.5))
```

سپس لایه های fully connected به شبکه اضافه میشوند. این لایه ها شامل لایه با ۳۲ نرون و سپس لایه dropout با احتمال ۰,۵ میباشد.

```
#softmax
model.add(keras.layers.Dense(2,activation='softmax'))
```

سپس لایه خروجی با ۲ نرون و تابع فعالسازی softmax را به شبکه اضافه میکنیم.

```
#compile
optimiser = keras.optimizers.Adam(learning_rate)
model.compile(optimizer = optimiser, loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

در انتها مدل را با بهینه ساز Adam نهایی میکنیم. در این مدل از تابع زیان sparse_categorical_crossentropy استفاده شده است که در اصل همان تابع cross entropy میباشد با این تفاوت که خروجی در مسئله ما عدد کلاس را نشان میدهد و به صورت one-hot نمیباشد (۰ نشان دهنده کلاس false و ۱ نشان دهنده کلاس true).

```
model.fit(X_train, Y_train, epochs = 10, batch_size = 32, validation_data = (X_val,Y_val))
```

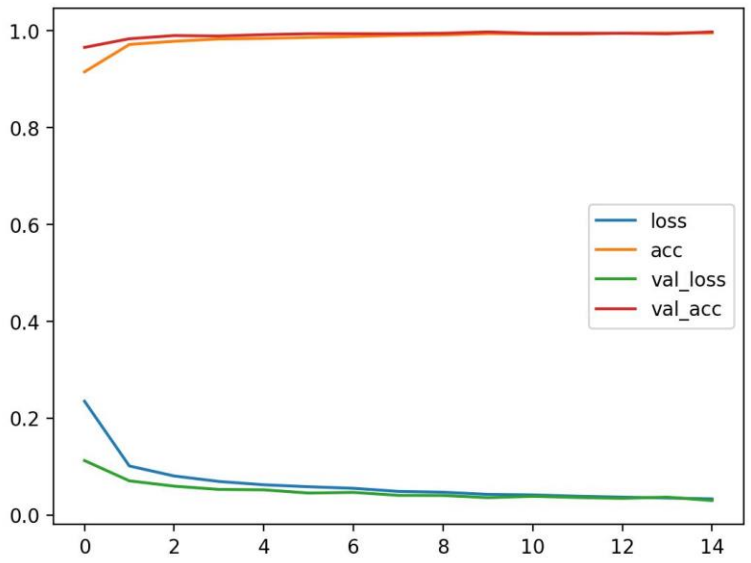
پس از مشخص کردن جزئیات شبکه عصبی به آموزش آن میپردازیم. برای اینکار از تابع fit استفاده میکنیم. برای آموزش از الگوریتم mini-batch gradient descent که الگوریتم پیش فرض تابع fit میباشد استفاده شده

است. تعداد تکرار بر روی کل داده ها (epoch) برابر با ۱۰ قرار گرفته است. تعداد اعضای هر batch در الگوریتم mini-batch برابر با ۳۲ نمونه قرار گرفته است.

نتایج به دست آمده از آموزش شبکه عصبی کانوولوشنال:

```
test error: 0.03737871228250425
test accuracy: 0.99323756
```

همانطور که در تصویر بالا مشاهده میشود، دقت مدل برابر با عدد ۰,۹۹ میباشد که این عدد به علت نوع مسئله میباشد و این مدل از مشکلات overfitting به اندازه مدل قبلی رنج نمیبرد و برای داده های جدید به خوبی عمل میکند.



در تصویر بالا روند تغییر دقت مدل در هر epoch نشان داده شده است.

برای پیشبینی خروجی توسط مدل ها کد predict.py پیاده سازی شد که در ادامه به توضیح آن میپردازیم.

```
#load the model
model = keras.models.load_model('E:\\project\\data2\\test2\\1\\model.h5')
```

ابتدا مدل را با استفاده از تابع load_model در یک متغیر ذخیره میکنیم.

```
def predict(filepath):
    signal, sr = librosa.load(filepath)

    if len(signal) != 22050:
        return

    signal = signal[:22050]
    mfcc = librosa.feature.mfcc(y=signal,sr=22050,n_mfcc=13,n_fft=2048,hop_length=512).T
```

برای پیشبینی بر روی یک فایل صوتی تابع predict پیاده سازی شده است. ورودی این تابع آدرس این فایل روی سیستم میباشد. ابتدا سیگنال صوتی را از روی فایل میخوانیم. سپس باید ویژگی mfcc را از این سیگنال استخراج کنیم چراکه مدل ما به این نوع ورودی برای پیشبینی خروجی نیاز دارند. اینکار را با تابع mfcc از کتابخانه ی librosa انجام میدهیم.

```
#  (# of samples, # of segments, # of coefs, # of channels)
mfcc = mfcc[np.newaxis, ...,np.newaxis ]
```

سپس ابعاد ورودی را برای استفاده ی مدل تغییر میدهیم.

```
preds = model.predict(mfcc)
idx = np.argmax(preds)
print('predicted keyword is ',end='')
print(mapping[idx])
```

سپس توسط تابع predict از مدل بر روی ویژگی های بدست آمده پیشبینی انجام میدهیم. خروجی مدل به صورت یک آرایه با دو درایه میباشد که درایه اول آن احتمال اینکه داده ورودی عضو کلاس 0 (کلاس false) را نشان میدهد و درایه دوم احتمال عضویت در کلاس 1 (کلاس true) را نشان میدهد. توسط تابع argmax اندیس احتمال بیشتر را بدست می آوریم و آن را در خروجی چاپ میکنیم.

۵-۳ راه حل های استفاده شده برای overfitting

همانطور که پیشتر اشاره شد مدل اول که پیاده سازی شد مشکل overfitting داشت به این معنا که تنها داده های مجموعه دادگان آموزش را به خوبی یاد گرفته بود و به عبارتی آنها را حفظ کرده بود. برای حل این مشکل چندین اقدام انجام گرفت. اولین اقدام اضافه کردن دیتا به دیتاست بود. برای اینکار کد record.py پیاده سازی شد که در قسمت های قبلی توضیح داده شد. از دیگر اقدامات میتوان به ساده کردن معماری مدل اشاره کرد که در قسمت پیاده سازی مدل توضیح داده شد. برای اینکار تعداد لایه های شبکه کم شدند و تعداد فیلترهای هر لایه نیز کاهش یافتند. از دیگر اقدامات بالا بردن احتمال dropout در شبکه عصبی بود. این لایه باعث خاموش شدن نرون ها در هر epoch میشود که این امر از تغییر وزن های این نرون ها جلوگیری میکند و از حفظ کردن داده ها توسط مدل و overfit شدن جلوگیری میکند. از دیگر اقدامات میتوان به کم کردن تعداد epoch ها اشاره

کرد. این امر باعث کمتر دیده شدن داده ها توسط شبکه میشود که باعث جلوگیری از حفظ کردن آنها میشود. یکی دیگر از مشکلات مدل تشخیص نویز و صدای محیط به عنوان کلمه کلیدی بود که برای جلوگیری از این موضوع صوت هایی با محتوای صدای محیط و نویز به صورت دستی ضبط و به مجموعه دادگان اضافه شدند.

فصل ششم

۶-۱ خلاصه

به طور خلاصه میتوان این پروژه را به دو بخش کلی یادگیری ماشین و برنامه دسکتاپ تقسیم کرد که بخش یادگیری ماشین پروژه بر عهده اینجانب بود.

کار های انجام شده در این بخش به همان ترتیبی که فصول این گزارش آورده شده است بودند که شامل جمع آوری داده ها، آماده سازی داده ها برای مدل های یادگیری عمیق و در نهایت انتخاب جزئیات شبکه های عصبی و پیاده سازی آنها برای استفاده در برنامه بود.

۶-۲ نتیجه گیری

امروزه پردازش صوت در محصولات شرکت های مختلف نقش مهمی را بازی میکند و در بسیاری موارد میتواند کاربرد داشته باشد. با پیشرفت های صورت گرفته در حوزه هوش مصنوعی و یادگیری عمیق و با جمع آوری داده ها به صورت هدفمند، عملکرد مدل ها بسیار بهبود یافته و در مواردی از انسان هم بهتر عمل کرده است. با استفاده از متد های یادگیری عمیق و در دسترس داشتن مقدار داده به اندازه کافی میتوان به مدل هایی با عملکرد بسیار خوب دست یافت که در این پروژه سعی شد تا یک نمونه از کاربرد های پردازش صوت پیاده سازی شود و به عملکرد مطلوبی هم دست یافتیم.

مراجع

[1] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuoyiin Chang, Tara Sainath. Deep Learning for Audio Signal Processing.

[2] Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aaron van den Oord, Sander Dieleman, Koray kavukeuoglu. Efficient Neural Audio Synthesis

[3] Yefei Chen, Heinrich Dinkel, Mengyue Wu, Kai Yu. Voice activity detection in the wild via weakly supervised sound event detection

[4] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, Zhenyao Zhu. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin

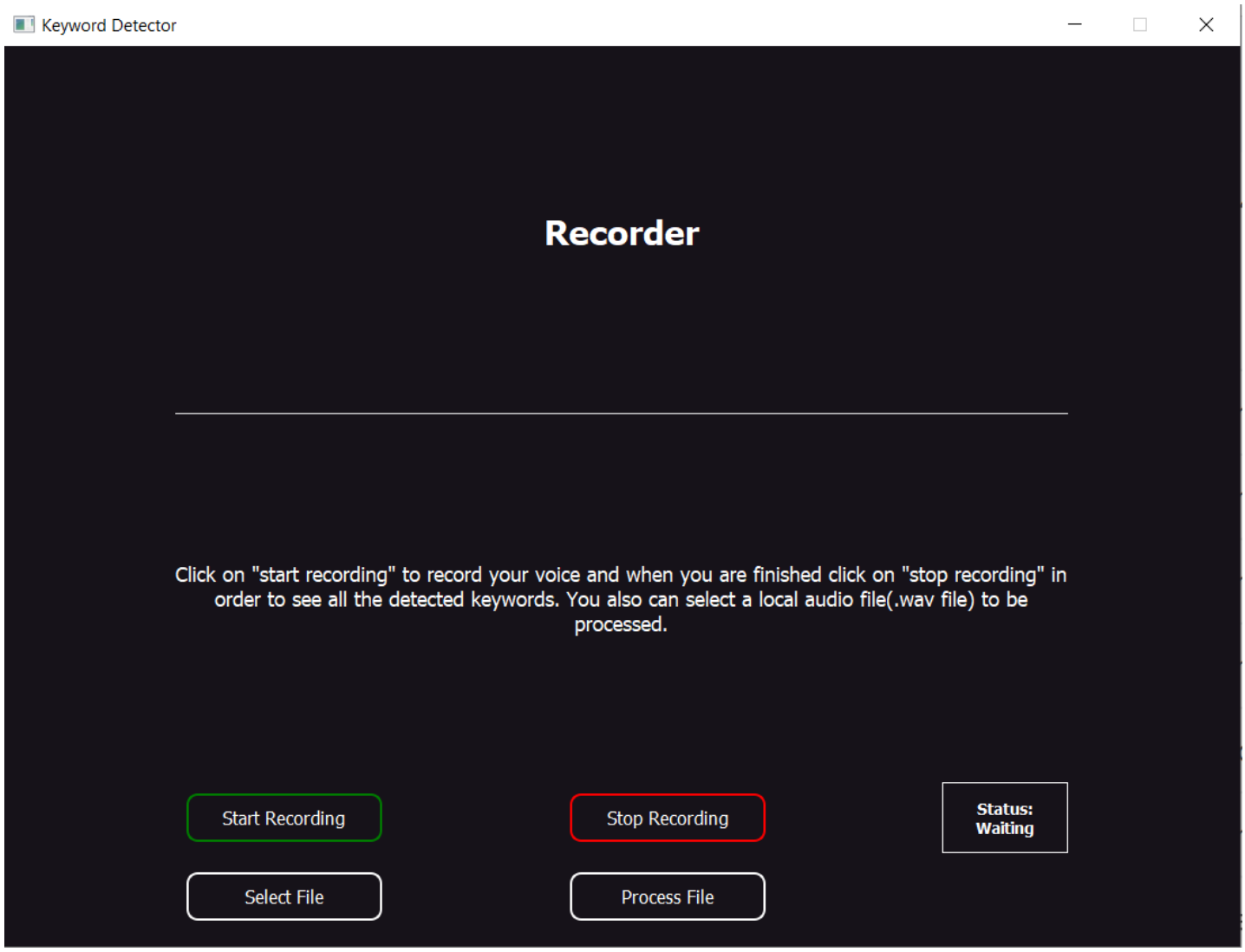
[5] Yun-Chia Liang, Iven Wijaya, Ming-Tao Yang, Josue Rodolfo Cuevas Juarez, Hou-Tai Chang. Deep Learning for Infant Cry Recognition

[6] Francois Chollet. Deep Learning with Python

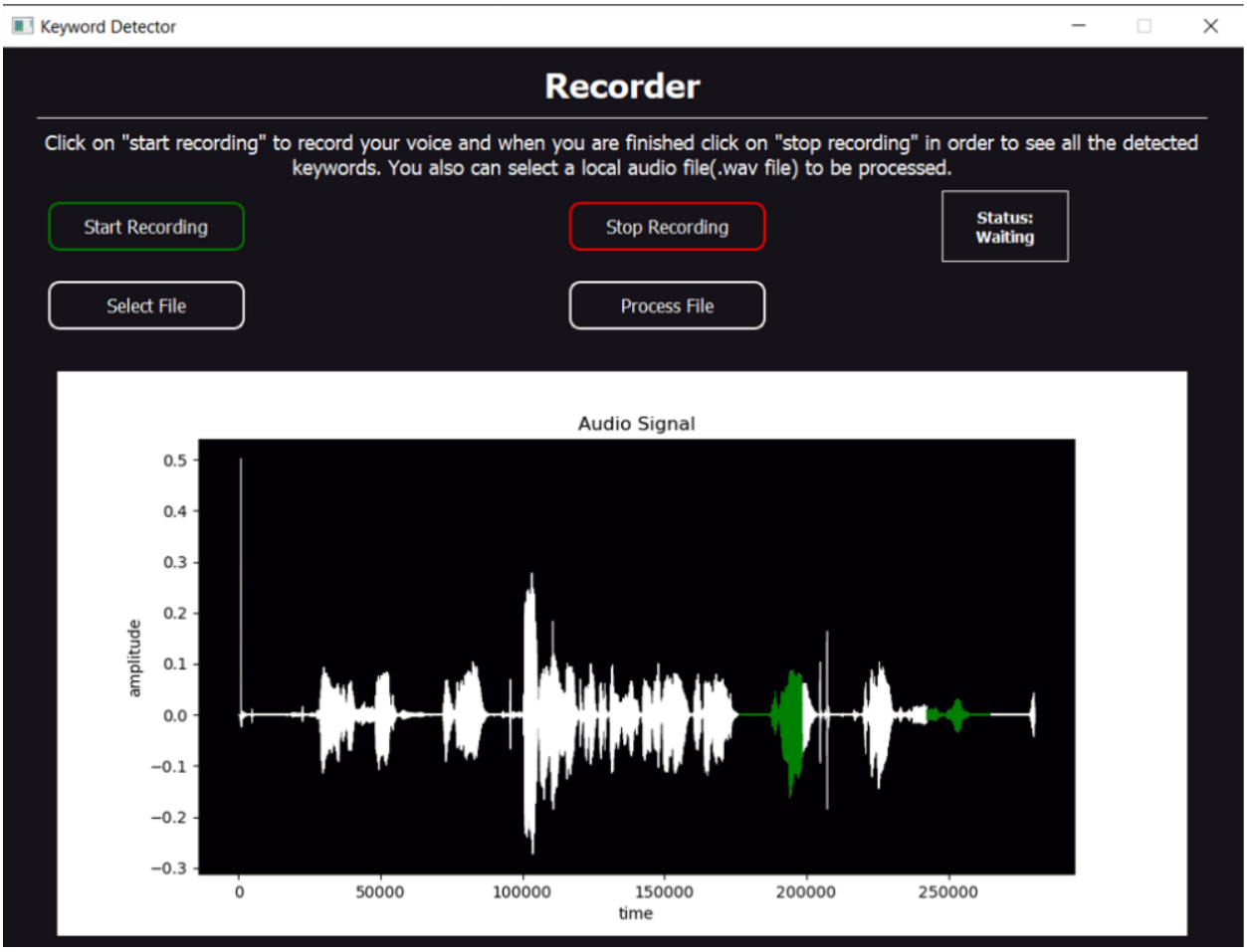
پیوست

نحوه کار با برنامه دسکتاپ

ابتدا کد app.py را اجرا میکنیم تا رابط کاربری گرافیکی نمایش داده شود.



برای استفاده از قابلیت تشخیص صوت در صوت ضبط شده ابتدا بر روی دکمه Start Recording کلیک میکنیم و زمانی که status به Listening تبدیل شد به این معنی است که برنامه در حال دریافت صدا از میکروفون میباشد. هر زمان که خواستیم تا ضبط متوقف شود بر روی Stop Recording کلیک میکنیم و منتظر خروجی برنامه میمانیم. خروجی برنامه به صورت تصویر زیر به نمایش گذاشته میشود:

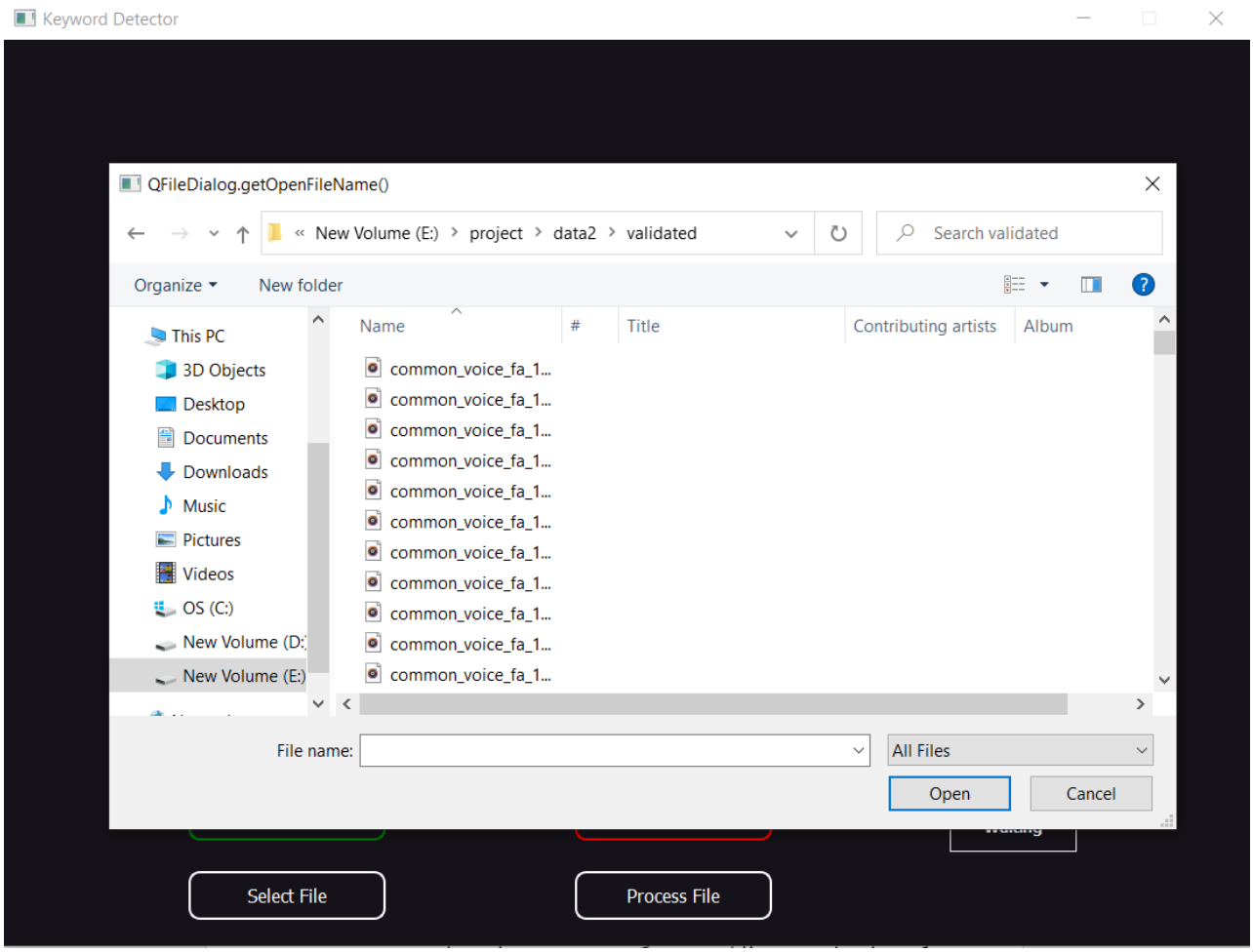


که در آن سیگنال صوت ضبط شده نمایش داده میشود و قسمت های سبز نشان دهنده ی بخش هایی از صوت است که برنامه کلمه کلیدی را تشخیص

داده است. همچنین در فولدری با نام detected فایل های صوتی ۱ ثانیه ای از بخش هایی که کلمه کلیدی تشخیص داده شده قرار میگیرد.

برای استفاده از قابلیت انجام پردازش بر روی یک فایل داخل سیستم خود، روی Select File کلیک میکنیم و فایل مورد نظر خود را انتخاب میکنیم. زمانی که status به Processing تبدیل شد بر روی Process File کلیک کرده تا پردازش روی فایل صورت بگیرد و خروجی نمایش داده شود. خروجی این بخش هم مانند بخش قبل میباشد.

در تصویر زیر قسمت انتخاب فایل برای این قابلیت را مشاهده میکنید:



University of Tehran

College of Farabi

Faculty of Engineering

Department of Computer Engineering

Implementation of a Persian Speech recognition System using deep learning methods

By:

Mohammadreza Afshari

Under Supervision of:

Dr. Zahra Movahedi

A Project Report as a Requirement for the Degree of Bachelor of Science in Computer Engineering

September 2022