



دانشگاه تهران
دانشکده‌گان فارابی
دانشکده مهندسی
گروه مهندسی کامپیوتر

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

نگارش:

محمدرضا افشاری

استاد راهنما:

دکتر زهرا موحدی

گزارش پروژه برای دریافت درجه ی کارشناسی در رشته مهندسی کامپیوتر

شهریور ۱۴۰۱

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

چکیده

امروزه با پیشرفت تکنولوژی و فناوری، زندگی روزمره به سمت دیجیتالی شدن پیش میرود. دستیار صوتی هوشمند یکی از فناوری‌های نسبتاً جدید است که کار را برای کنترل ابزارهای مختلف مانند تلفن همراه هوشمند، تبلت، لپ‌تاپ و حتی تلویزیون و برخی دیگر از دستگاه‌های الکترونیکی آسان‌تر میکند.

تنها کافی‌ست نیازها و خواسته‌های خود را بیان کنیم تا دستیار صوتی هوشمند به دستورهای ما عمل کند. بنابراین این برنامه با گوش کردن صدای شما، هرآنچه را که خواسته شود انجام میدهد.

در این پروژه، با استفاده از دیتاست‌های موجود در اینترنت (در قسمت ۴-۱ تمامی دیتاست‌های استفاده شده توضیح داده شده‌اند) و جمع‌آوری دیتا و استخراج ویژگی‌های مورد نیاز از سیگنال‌های صوتی و استفاده از مدل‌های یادگیری عمیق CNN (شبکه عصبی کانوولوشنال) و RNN (شبکه عصبی بازگشتی)، مدلی برای تشخیص کلمات کلیدی در صوت را ایجاد کردیم که قادر به تشخیص کلمات و مشخص کردن محل رخداد آن در صوت میباشد.

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

کلمات کلیدی: بازشناسی الگو، پردازش گفتار، یادگیری عمیق، سیگنال صوتی، شبکه عصبی کانوولوشنال، شبکه عصبی بازگشتی، برنامه دسکتاپ

فهرست

۱ فصل اول

۸..... ۱-۱ مقدمه

۲ فصل دوم

۱۰..... ۱-۲ مروری بر کارهای مشابه

۳ فصل سوم

۱۱..... ۱-۳ شرح کلی سیستم

۱۲..... ۲-۳ بررسی الگوریتم‌های یادگیری عمیق

۱۲..... ۱-۲-۳ شبکه‌های عصبی مصنوعی

۱۷..... ۲-۲-۳ شبکه‌های عصبی کانوولوشنال

۲۱..... ۳-۲-۳ شبکه‌های عصبی بازگشتی

۲۵..... ۳-۳ معیارهای سنجش عملکرد مدل‌های یادگیری ماشین

۲۶..... ۴-۳ ابزارها

۴ فصل چهارم

۲۸..... ۱-۴ جمع‌آوری داده‌ها

۲۸..... ۱-۱-۴ جمع‌آوری دستی داده‌ها

۳۰..... ۲-۴ بررسی کلی داده‌ها

۳۳..... ۳-۴ استخراج و جداسازی کلمات کلیدی

۳۷..... ۴-۴ ویژگی‌های قابل استخراج و استفاده برای صوت

۳۷..... ۱-۴-۴ سیگنال صوتی خام

۳۸..... zero-crossing rate ۲-۴-۴

۳۹..... Root-mean-square energy ۳-۴-۴

- ۴۰..... spectrogram ۴-۴-۴
- ۴۱..... mel frequency cepstral coefficients ۵-۴-۴
- ۴۲..... ایجاد دیتاست ۵-۴
- ۴۵..... راه حل های موجود برای دیتاست نامتوازن ۶-۴
- ۴۵..... upsampling ۱-۶-۴
- ۴۸..... downsampling ۲-۶-۴

۵ فصل پنجم

- ۴۹..... ۱-۵ ارائه معماری برای مدل ها
- ۴۹..... ۱-۱-۵ مدل CNN
- ۵۰..... ۲-۵ پیاده سازی مدل
- ۵۶..... ۳-۵ راه حل های استفاده شده برای overfitting
- ۵۶..... ۴-۵ مقایسه عملکرد مدل ها

۶ فصل ششم

- ۵۷..... ۱-۶ خلاصه
- ۵۷..... ۲-۶ نتیجه گیری
- ۵۸..... ۳-۶ مراجع
- ۵۹..... ۴-۶ پیوست - نحوه کار با برنامه

فصل اول

۱-۱ مقدمه

مهندسان و دانشمندان پدیده ارتباط گفتاری را با چشم‌اندازی به ایجاد سیستم‌های کارآمدتر و مؤثرتر ارتباط با انسان و انسان با ماشین مطالعه کرده‌اند. 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 دهه‌ها تحقیق به دست آمده است. به زبان دیگر پردازش گفتار به توانایی یک ماشین یا برنامه برای شناسایی کلماتی که انسان‌ها صحبت میکنند گفته میشود.

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

از کاربردهای پردازش صوت و گفتار میتوان به دستیارهای صوتی، خانه‌های هوشمند، وسایل الکترونیکی که قادر به تشخیص چند دستور محدود هستند و ... اشاره کرد.

از انواع این سیستم‌ها به سیستم تشخیص کلمه‌کلیدی برای کاربردهای محدود (برای مثال تشخیص کلمه خاص برای روشن شدن دستگاه یا کنترل دستگاه با صوت)، دستیارهای صوتی با کاربردهای مختلف، تبدیل صوت به متن، ترجمه‌ی همزمان گفتار اشاره کرد که این سیستم‌ها در زندگی امروزه نقش بسزایی را بازی میکنند.

با پیشرفت‌های صورت گرفته در حوزه هوش مصنوعی و یادگیری عمیق، این سیستم‌ها به عملکردی نزدیک و حتی بالاتر و بهینه‌تر از انسان‌ها دست یافته‌اند و با زیاد شدن دیتا و جمع آوری هدف دار آنها توسط شرکت ها، این پیشرفت‌ها روز به روز چشمگیرتر میشوند.

از چالش‌های این سیستم‌ها میتوان به جمع‌آوری داده‌های مناسب و قابل استفاده برای این مسائل اشاره کرد. از دیگر چالش‌ها میتوان به بحث تشخیص اشتباه کلمه‌کلیدی و یا تشخیص ندادن آن اشاره کرد. در برخی موارد این مشکل‌ها میتواند روی تجربه‌ی کاربرها تاثیر منفی بگذارد اما در موارد جدی‌تر و حیاتی میتواند باعث بروز مشکل‌های حادثتری شود. برای مثال در سیستم‌های دستیار صوتی اگر کلمه‌ای که برای فعال کردن این سیستم‌ها مورد استفاده قرار میگیرد تشخیص داده نشود صرفاً در تجربه کاربر تاثیر منفی میگذارد اما برای مثال در سیستمی که وظیفه تشخیص صدای شلیک را دارد، این موضوع تشخیص ندادن میتواند تاثیرهای جدی‌تری داشته باشد. به طور کلی باید با توجه به مساله این دو موضوع کنترل شوند.

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

۱. جمع آوری داده‌ها – جمع آوری داده‌های خام موجود در اینترنت (بصورت کامل در بخش ۴-۱ در رابطه با مجموعه دادگان استفاده‌شده توضیح داده‌شده‌است) و جمع آوری آنها به صورت دستی
۲. تبدیل داده‌های خام به داده‌های قابل استفاده – پاکسازی و استخراج بخش‌های مفید دیتاست‌ها و لیبیل زدن آنها
۳. پیش‌پردازش داده‌ها و ایجاد دیتاست – تبدیل داده‌ها به فرمت قابل استفاده برای مدل‌های یادگیری عمیق و جمع‌آوری آنها در یک دیتاست واحد برای سهولت در فاز train مدل

۴. مشخص کردن معماری مدل‌ها – مشخص کردن جزئیات مدل‌ها

۵. پیاده‌سازی مدل‌ها و train آنها

۶. بهینه‌سازی عملکرد مدل‌ها

۷. استفاده از چند نخی برای کار با مدل در کنار برنامه اصلی

۸. پیاده‌سازی برنامه دسکتاپ

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

فصل دوم

۱-۲ مروری بر کارهای مشابه

با پیشرفت هوش مصنوعی و یادگیری عمیق و همچنین بالا رفتن توانایی پردازش سخت افزارها و بهینه‌سازی الگوریتم‌های یادگیری ماشین، پیشرفت‌های زیادی در زمینه پردازش صوت صورت گرفته است. در ادامه مقالاتی که در این زمینه کار شده آورده شده است.

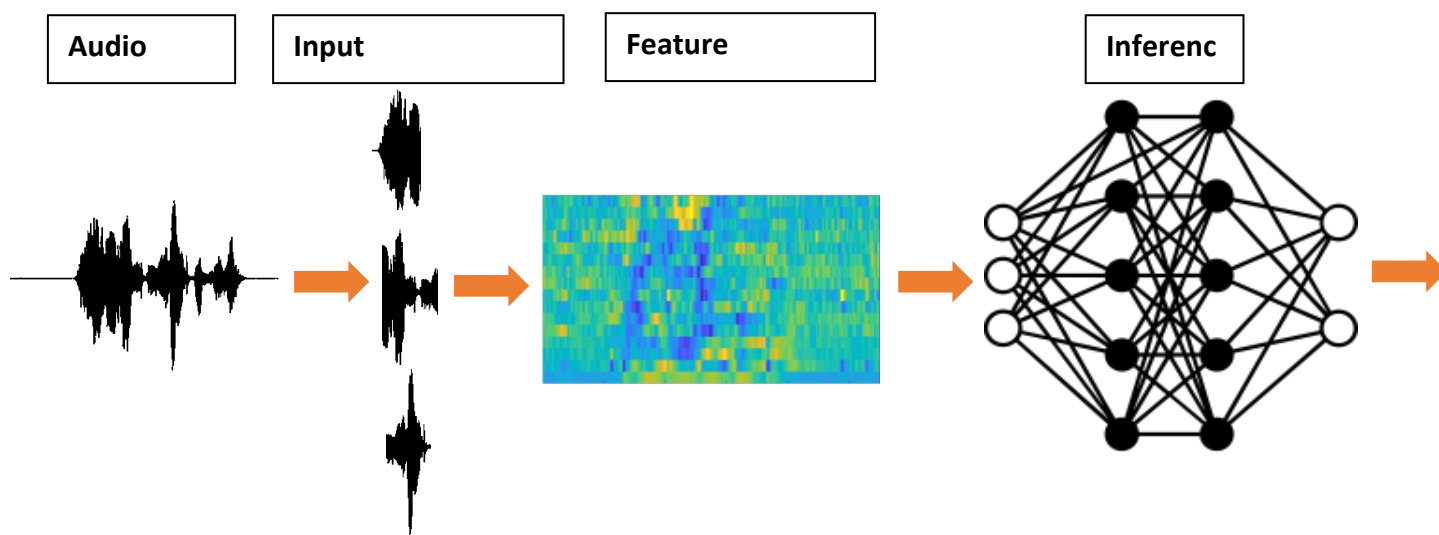
ردیف	شماره مرجع	محتویات پژوهش
۱	[1]	مروری بر روش‌ها و پیشرفت‌های یادگیری عمیق در زمینه‌ی پردازش صوت و بررسی تکنیک‌های به روز استفاده شده در دنیای واقعی
۲	[2]	ارائه روش بهینه برای سنتز صوت و تبدیل متن به صوت با استفاده از روش‌های یادگیری عمیق و مدل‌های شبکه عصبی بازگشتی
۳	[3]	ارائه مدلی برای تشخیص فعالیت 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 موارد دیگر میتوان به دستیارهای 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 شرکت‌ها 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 تلفن‌های هوشمند میشوند. از دیگر موارد استفاده از پردازش صوت میتوان به دستیارهای صوتی برای افراد نابینا اشاره کرد که با خواندن اطلاعات روی صفحه نمایش تلفن همراه یا سیستم‌های خانگی با لحنی نزدیک به انسان، امکان استفاده از این محصولات را به این افراد میدهند. از دیگر موارد استفاده سیستم‌های تشخیص wake word در لوازم الکترونیکی میباشد. برای مثال در اکثر کنسول‌های بازی امکان روشن شدن این کنسول‌ها 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 الگوریتم‌های مورد استفاده میپردازیم. در سیستم پیاده‌سازی شده این پروژه، داده‌های ورودی از طریق میکروفون یا بصورت فایل‌های صوتی به سیستم داده‌میشود و سپس یک سری اقدامات بر روی این داده‌ی ورودی انجام میگردد تا این داده‌ها قابل استفاده توسط مدل‌های ما شوند. در این بخش ابتدا سیگنال صوتی به بخش‌هایی با اندازه مساوی تقسیم‌میشود (در این پروژه به قسمت‌های ۱ ثانیه ای). این تقسیم‌بندی به صورتی انجام میشود که قسمت‌ها با هم اشتراک داشته باشند تا اگر کلمه مورد نظر مدل در این صوت گفته‌شده‌بود در این قسمت‌بندی 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 خواهیم آورد. بعد از این تقسیم‌بندی، ویژگی‌ها از این قسمت‌ها استخراج میشوند و آماده‌ی استفاده برای مدل‌ها میشوند. سپس این ویژگی‌های استخراج‌شده به عنوان ورودی به مدل داده میشود تا مدل پیشبینی را انجام دهد. پس از آن خروجی به کاربر به نمایش در آورده میشود.



تصویر شماره ۱: System's Pipeline

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

- کد `convert.py` برای تبدیل فایل‌های با فرمت `mp3` به `wav`. تا قابل پردازش توسط کتابخانه‌ها و پایتون باشند. فصل ۴
- کد `label.py` برای لیبل زدن فایل‌های صوتی. فصل ۴
- کد `data_prep.py` برای بررسی کلی داده‌ها. فصل ۴
- کد `extract_keys.py` برای جداسازی فایل‌هایی که حاوی کلمه کلیدی هستند. فصل ۴
- کد `Record.py` برای ضبط صدا و جمع کردن دستی دیتا. فصل ۴

- کد model.py برای پیاده‌سازی مدل‌ها. فصل ۵
- کد predict.py برای انجام پیش‌بینی‌ها توسط مدل‌ها. فصل ۵
- کد upsample.py برای رفع مشکل نامتوازن بودن دیتاست‌ها. فصل ۴
- کد make_dataset.py برای ایجاد دیتاست‌های قابل استفاده توسط مدل‌ها. فصل ۴

۲-۳ بررسی الگوریتم‌های یادگیری عمیق

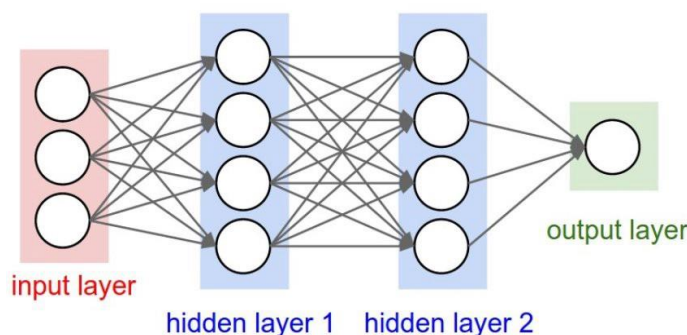
در این بخش به بررسی الگوریتم‌های مختلف یادگیری عمیق و استفاده شده در این پروژه خواهیم پرداخت.

در این پروژه از الگوریتم‌های یادگیری عمیق برای ایجاد مدل‌ها استفاده شده. از جمله این مدل‌ها شبکه عصبی کانولوشنال و شبکه عصبی بازگشتی هستند که در این پروژه مورد استفاده قرار گرفته اند.

یادگیری عمیق یک زیرشاخه از یادگیری ماشین و بر مبنای مجموعه‌ای از الگوریتم‌ها 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 یاخته‌های عصبی با فرستادن پیام به یکدیگر درک را امکان پذیر میکنند، بسته به فرض‌های گوناگون در مورد نحوه اتصال این یاخته‌های عصبی، مدل‌ها و ساختارهای مختلفی در این حوزه پیشنهاد و بررسی شده‌اند، هرچند که این مدل‌ها به صورت طبیعی در مغز انسان وجود ندارد و مغز انسان پیچیدگی‌های بیشتری را داراست. از جمله این مدل‌ها میتوان به مدل‌های شبکه عصبی عمیق، شبکه عصبی کانولوشنال و شبکه عصبی بازگشتی اشاره کرد.

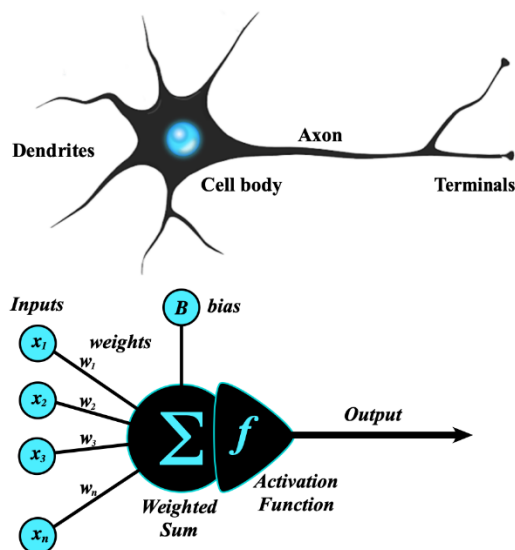
۳-۲-۱ شبکه‌های عصبی مصنوعی

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



تصویر شماره ۲

این نرون‌ها از بخش‌های مختلفی تشکیل شده است که این بخش‌ها شامل تابع فعال‌سازی غیرخطی و وزن‌ها بر روی اتصالات ورودی و خروجی آن در قالب یک تبدیل خطی میشود.



تصویر شماره ۳

در این نرون یک بردار از وزن‌ها و یک مقدار bias نگهداری میشود که در نهایت پس از اعمال آنها روی ورودی‌های نرون، حاصل‌ها با یکدیگر جمع میشوند. این وزن‌ها اهمیت هر ورودی را نشان میدهند و تاثیر هر یک را روی خروجی این نرون مشخص میکنند. در پروسه آموزش شبکه عصبی مقادیر این وزن‌ها 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 وزن‌دار روی ورودی‌های نرون صورت میگیرد. این مجموع در نهایت با مقدار bias جمع میشود.

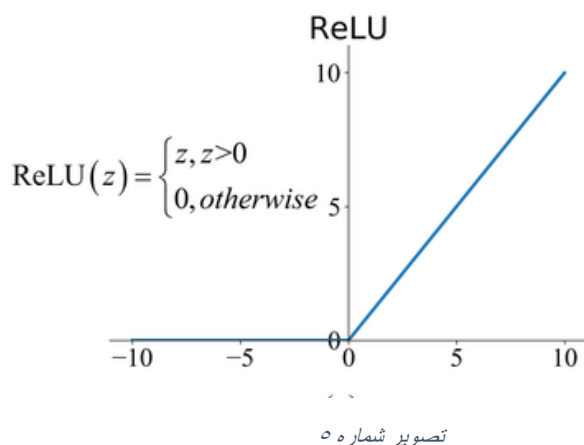
$$\sum_{j=1}^n x_j w_j$$

تصویر شماره ۴

در ادامه به بررسی توابع فعال‌سازی میپردازیم. استفاده از این توابع غیرخطی برای بازنمایی‌های پیچیده و الگوهای پیچیده داده‌ها لازم است چراکه بدون استفاده از این تابع‌ها تمامی نرون‌ها یک تبدیل خطی روی داده‌ها اعمال میکنند و کل شبکه به یک ترکیب خطی از ورودی‌ها تبدیل میشود و عملاً تنها الگوهای ساده و خطی قابل استخراج میشوند. این توابع مانند نرون‌های مغز فعال میشوند و اتصال خود را با نرون بعدی برقرار میکنند. از توابع فعال‌سازی متداول که در شبکه‌های 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.

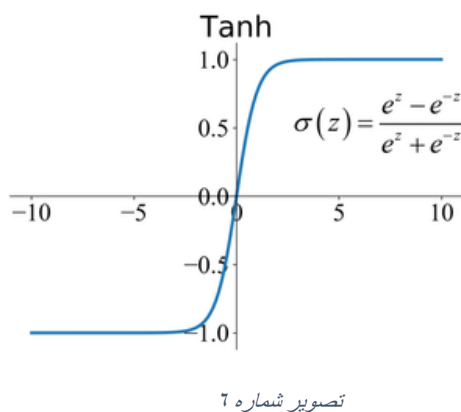
● تابع فعالسازی ReLU — Rectified Linear Unit activation function تابع یا به اختصار ReLU تابعی

ست با نمودار و فرمول زیر:



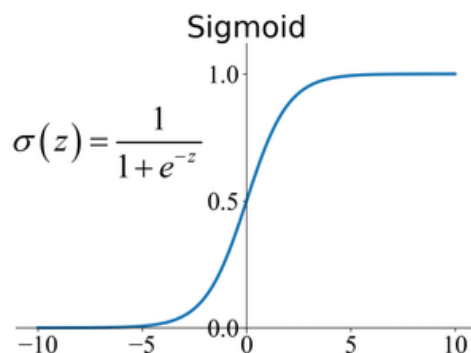
که خروجی آن برای ورودی‌های بزرگتر از ۰ برابر با همان ورودی است. این تابع متداول‌ترین تابع در شبکه‌های عصبی می‌باشد چراکه فرایند آموزش با این تابع سریع‌تر هست. این تابع از نظر محاسباتی نیز بسیار کارآمد است و سرعت همگرایی بالایی دارد. این تابع مشکل مرگ نرون را دارد که در مقادیر ورودی نزدیک صفر گرادیان 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 -۱ 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.

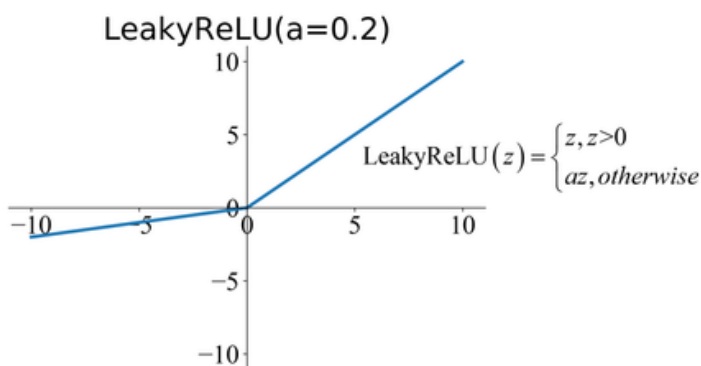
● تابع فعالسازی Sigmoid – تابعی با فرمول و نمودار زیر:



تصویر شماره ۷

این تابع یک منحنی S شکل است. زمانی که 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 همگرایی کند را دارد.

● تابع leaky ReLU – تابعی با فرمول و نمودار زیر:



تصویر شماره ۸

انتخاب تابع فعالسازی برای نرون‌ها 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 کلاس‌بندی 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 لایه‌های 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 وزن‌های 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 ورودی‌های مدل (مجموعه داده‌های آموزش یا training data) محاسبه میشود و با توجه به این خطا وزن‌ها تنظیم میشوند. یکی از روش‌هایی 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 اول که حرکت رو به جلو یا feed forward نام دارد، داده‌های ورودی از لایه‌ی ورودی به شبکه داده میشوند و این داده‌ها 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، وزن‌ها به هنگامسازی میشوند. این به هنگامسازی با توجه به مقدار تابع خطا صورت میگیرد تا در تکرارهای 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 گرادیان‌ها 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 محاسبه‌ی گرادیان تابع خطا وزن‌ها 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 شبکه‌های عصبی وجود دارند که میتوان چند نمونه از آنها را در ادامه مشاهده کرد (در فرمول‌هایی که در ادامه آورده شده است، \hat{y}_i نشان دهنده‌ی مقدار پیشبینی شده توسط شبکه و y_i نشان دهنده‌ی مقدار واقعی خروجی میباشد)

- **تابع Means Square Error –** یکی از متداول‌ترین توابع مورد استفاده در شبکه‌های 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:

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

تصویر شماره ۹

- **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:

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

تصویر شماره ۱۰

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

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

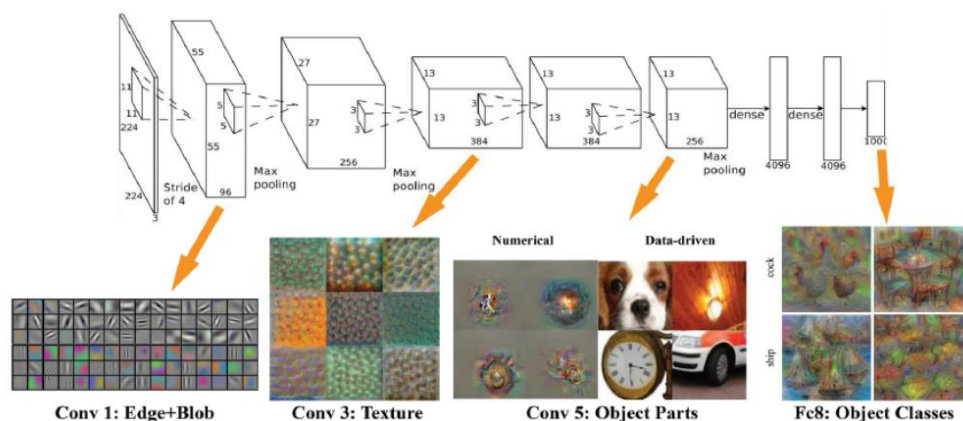
تصویر شماره ۱۱

در نهایت با استفاده از این توابع مقدار کلی خطا برای تمامی داده‌های آموزش با میانگین‌گیری روی تمام خطاها محاسبه میشود. در نهایت با استفاده از روش گرادیان کاهشی و تکرار بر روی داده‌های آموزش، مقادیر وزن‌ها برای مینیمم‌سازی میزان خطا بهینه میشوند تا مدل با خطای کمی خروجی را پیشبینی کند.

۳-۲-۲ شبکه‌های عصبی کانوولوشنال

یکی از مدل‌های رایج شبکه‌های 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 شبکه‌ها یک بازنمایی پیچیده از داده‌های تصویری را با استفاده از مقدار حجیمی از داده‌ها یاد میگیرند. این شبکه‌ها از سیستم بینایی انسان الهام گرفته شده‌اند و چندین لایه از تبدیلات را یاد میگیرند. این شبکه‌ها از دو نوع لایه تشکیل میشوند که به شرح زیر میباشد:

- **لایه کانوولوشن** – این لایه با توجه به اینکه در تصاویر پیکسل‌ها 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، ویژگی‌ها را استخراج میکند.
- **لایه تلفیق** – در این لایه ماکزیمم یا متوسط ویژگی‌ها بر روی یک ناحیه خاص محاسبه میشوند تا اندازه‌ی تصویر کاهش یابد و محاسبات در لایه‌های بعدی سبک‌تر شود.

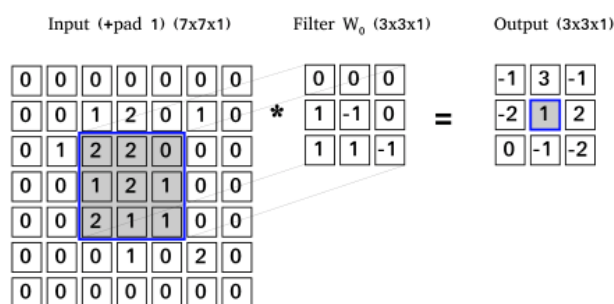


تصویر شماره ۱۲

هر لایه‌ی کانولوشن از تعدادی فیلتر تشکیل شده است که هر یک از این فیلترها به صورت یک ماتریس دوبعدی از اعداد هستند که در اصل معادل با وزن‌ها در شبکه عصبی مصنوعی عمل میکنند و در نهایت هدف تنظیم این مقادیر برای بهینه‌سازی عملکرد شبکه میباشد.

عملیات کانولوشن:

در این عملیات یک ماتریس دوبعدی به عنوان فیلتر بر روی مقادیر پیکسل‌های 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 وزن‌های فیلتر به عملکردی بهتر و بهینه‌تر از نظر محاسباتی رسید. در تصویر زیر میتوان نحوه‌ی عملکرد این عملیات را مشاهده کرد:



تصویر شماره ۱۳

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 بخش‌های تصویر را پوشش دهد. نحوه‌ی محاسبات به این صورت

میباشد که هر درایه از ماتریس در درایه متناظر با آن در فیلتر ضرب میشود و در نهایت نتیجه‌ی تمامی اینها با هم جمع شده و به عنوان خروجی در نظر گرفته میشود. برای مثال برای تصویر بالا داریم:

$$2 \times 0 + 2 \times 0 + 0 \times 0 + 1 \times 1 + 2 \times -1 + 1 \times 0 + 2 \times 1 + 1 \times 1 + 1 \times -1 = 1$$

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:

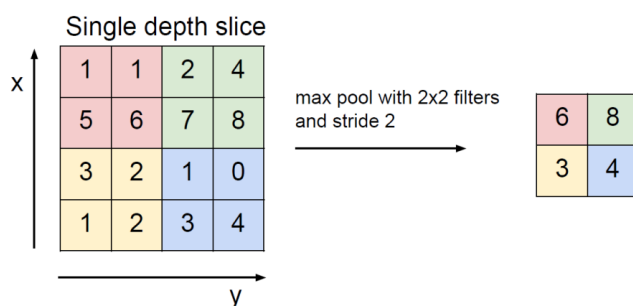
از این عملیات برای کاهش ابعاد بازنمایی‌ها استفاده میشود. روش‌های مختلفی برای این عملیات وجود دارد که در ادامه به آنها میپردازیم.

– 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:

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

تصویر شماره ۱۴

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



تصویر شماره ۱۵

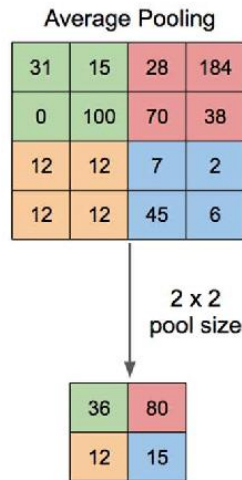
در تصویر بالا یک فیلتر max-pool ۲ در ۲ با اندازه گام ۲ بر روی تصویر اعمال شده که خروجی به صورت ماتریس سمت راست میشود.

– 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:

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

تصویر شماره ۱۶

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

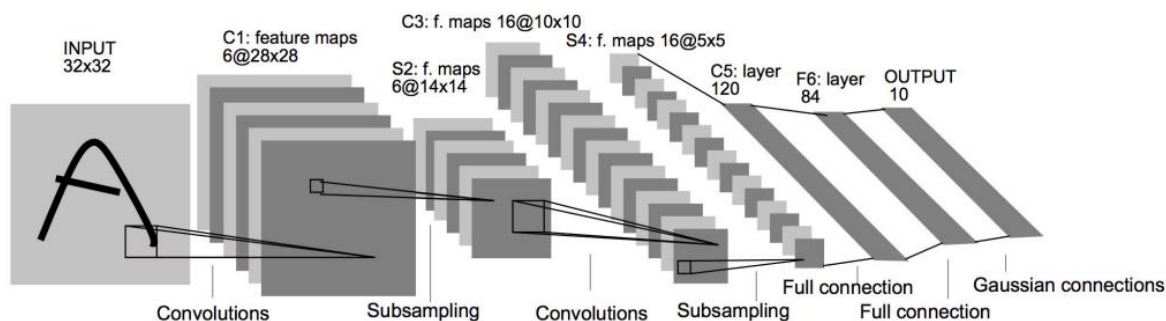


تصویر شماره ۱۷

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 متداولترین روشها میباشد و معمولا از یکی از این دو روش در شبکههای کانوولوشنال استفاده میشود.

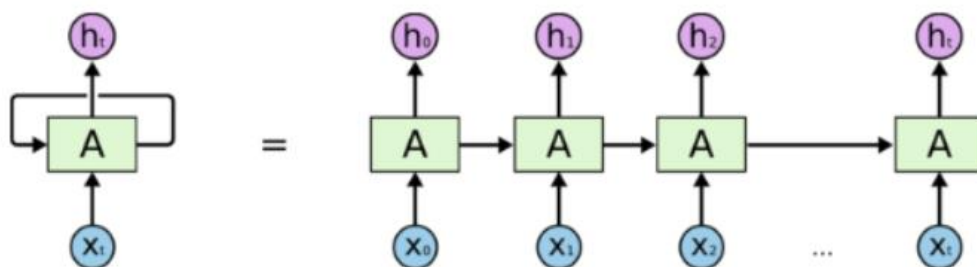
پس از لایههای کانوولوشن و تلفیق، برای بدست آوردن خروجی که به صورت تک عدد میباشد، نمیتوان از فیلترها که خروجیشان ماتریسی از اعداد است استفاده کرد و در لایههای آخر شبکههای کانوولوشنال از یک لایه‌ی تماما متصل (Fully Connected Layer) استفاده میشود. ورودی این لایه تخت شده‌ی ماتریس خروجی آخرین لایه‌ی کانوولوشن 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 شبکههای کانوولوشنال معمولا چندین لایه‌ی آخر شبکه لایههای Fully Connected میباشد که در لایه‌ی خروجی با توجه به نوع مسئله تابع فعالسازی مناسب انتخاب میشود. در تصویر زیر یک نمونه از شبکه عصبی کانوولوشنال را مشاهده میکنید:



تصویر شماره ۱۸

۳-۲-۳ شبکه‌های عصبی بازگشتی

شبکه عصبی بازگشتی Recurrent Neural Network یا به اختصار RNN نوعی از شبکه‌های عصبی میباشند که در تشخیص گفتار، پردازش زبان طبیعی، پردازش صوت و به صورت کلی پردازش داده‌های ترتیبی (Sequential data) استفاده میشود. در این شبکه‌ها برخلاف شبکه‌های معرفی شده تا اینجا، یک لایه‌ی بازخورد داریم که خروجی شبکه به همراه ورودی بعدی به شبکه بازگردانده میشود. این شبکه‌ها با استفاده از این مکانیزم دارای حافظه داخلی میباشند و میتوانند ورودی‌های قبلی را به خاطر بسپارند و از این حافظه برای پردازش دنباله‌ای از ورودی‌ها بهره ببرند.



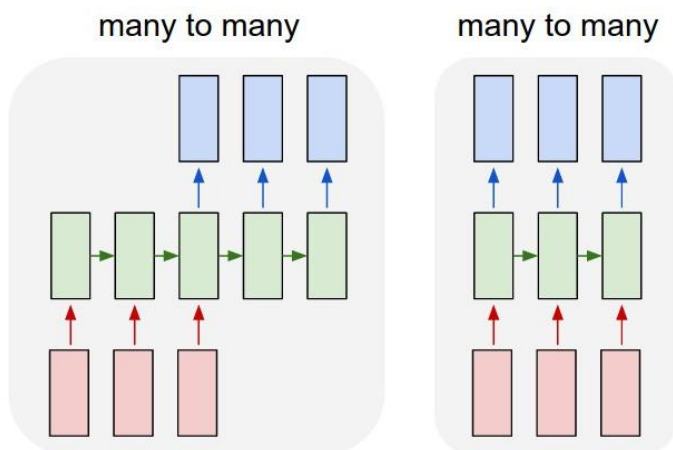
An unrolled recurrent neural network.

تصویر شماره ۱۹

در تصویر بالا یک 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 مکانیزمی میتواند اطلاعاتی از زمان‌های قبل‌تر هم نگه دارد.

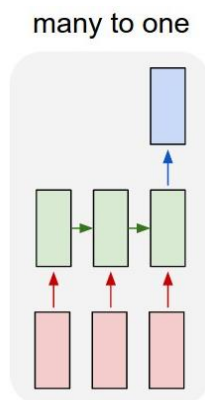
معماری‌های مختلفی از این شبکه وجود دارد که میتوان به نمونه‌های زیر اشاره کرد:

- 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.



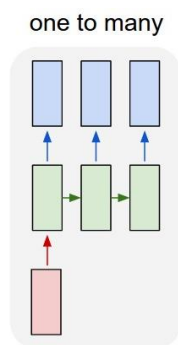
تصویر شماره ۲۰

- Many-to-one در این معماری ورودی به صورت یک دنباله و خروجی به صورت یک عدد میباشد. از موارد استفاده 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 جمله‌ی 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.



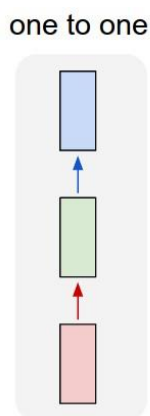
تصویر شماره ۲۱

- **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.



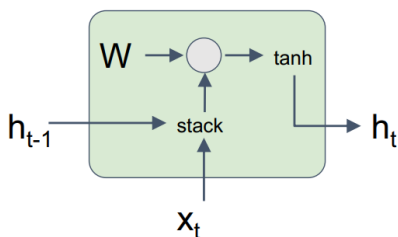
تصویر شماره ۲۲

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



تصویر شماره ۲۳

- در شبکه‌های عصبی بازگشتی نیز هر نرون حاوی وزن‌ها و **bias** و یک تابع فعالساز می‌باشد و تنها معماری آن متفاوت می‌باشد. در این مدل شبکه عصبی معمولاً از تابع فعالساز **tanh** استفاده می‌شود.
- یک واحد ساده از شبکه عصبی بازگشتی به صورت زیر می‌باشد:



تصویر شماره ۲۴

فرمول‌های مورد استفاده در این واحد به صورت زیر می‌باشد:

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

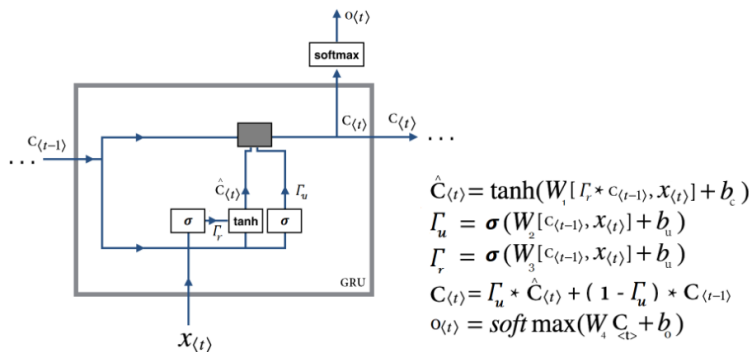
تصویر شماره ۲۵

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

این مدل ساده تنها از خروجی یک زمان قبل‌تر استفاده می‌کند و از مشکل محوشدگی گرادیان رنج می‌برند. برای استفاده از اطلاعات زمان‌های قبل‌تر باید به این واحد حافظه اضافه کنیم. دو مدل واحد متعارف حافظه‌دار با نام‌های LSTM و GRU وجود دارند.

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

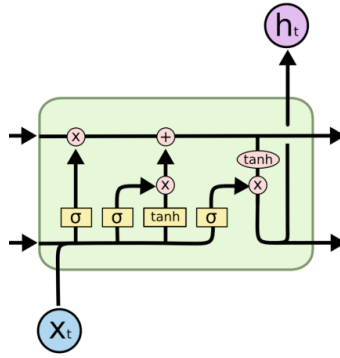
در تصویر زیر می‌توان یک واحد GRU و فرمول‌های مربوط به گیت‌های آن را مشاهده کرد:



تصویر شماره ۲۶

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 نشان Γ_u داده می‌شود و فرمول مربوط به گیت بازنشانی با نمایش Γ_r داده می‌شود. هر یک از این گیت‌ها وزن‌های مخصوص به خود را دارند که همگی در فرایند آموزش تنظیم می‌شوند.

LSTM یا Long Short Term Memory دیگر واحد حافظه دار برای شبکه‌های بازگشتی می‌باشد. این واحد هم مانند GRU با استفاده از گیت‌ها اطلاعات را از زمان‌های 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 فرمول‌های مربوط به آن را مشاهده کرد:



تصویر شماره ۲۷

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

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

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

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

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

تصویر شماره ۲۸

در فرمول‌های بالا فرمول اول مربوط به گیت فراموشی و فرمول دوم مربوط به گیت ورودی و فرمول سوم مربوط به حافظه 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 گام‌های زمانی قبلی در تصمیم‌گیری تاثیر دارند.

۳-۳ معیارهای سنجش عملکرد مدل‌های یادگیری ماشین

قبل از بررسی روش‌ها و معیارها باید به چند مورد که در فرمول‌های این روش‌ها استفاده میشود اشاره میکنیم.

- **True Positive:** داده‌هایی که برای کلاس True هستند (در این پروژه داده‌هایی که کلمه کلیدی هستند) و مدل به درستی کلاس آنها را تشخیص داده است.
- **False Positive:** داده‌هایی که برای کلاس False هستند (در این پروژه داده‌هایی که کلمه کلیدی نیستند) و مدل به اشتباه آنها را بعنوان کلاس True تشخیص داده است.

- **True Negative:** داده هایی که برای کلاس False هستند و مدل به درستی کلاس آنها را تشخیص داده است.
- **False Negative:** داده هایی که برای کلاس True هستند و مدل به اشتباه کلاس آنها را تشخیص داده است.

در ادامه به معیارهای ارزیابی مدل های یادگیری ماشین میپردازیم:

Precision: این معیار دقت مدل برای تشخیص کلاس ۱ را نشان میدهد. این مقدار دقت با تقسیم تعداد نمونه هایی که به درستی تشخیص داده شده اند بر تعداد کل نمونه ها بدست می آید. در تصویر زیر فرمول این معیار را مشاهده میکنید.

$$Precision = \frac{TP}{TP + FP}$$

تصویر شماره ۲۹

Recall: این معیار نسبت تعداد نمونه هایی از کلاس ۱ که مدل آنها را به درستی تشخیص داده را به کل نمونه های با کلاس ۱ نشان میدهد.

$$Recall = \frac{TP}{TP + FN}$$

تصویر شماره ۳۰

F1-score: این معیار میانگین هارمونیک از دو معیار قبلی را به ما میدهد که با رابطه زیر قابل محاسبه است.

$$F1\ score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

تصویر شماره ۳۱

Support: تعداد نمونه ها در هر کلاس را مشخص میکند.

۳-۴ ابزارها

در این بخش به معرفی ابزارهای 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 داده‌های صوتی میباشد و امکاناتی نظیر خواندن فایل‌های صوتی، استخراج ویژگی‌ها از سیگنال‌های صوتی، اعمال تغییرات و تبدیل‌های متداول بر روی سیگنال‌های صوتی، تصویرسازی انواع بازنمایی‌های صوت، ذخیره‌سازی خروجی فایل صوتی و دارد.
 - **کتابخانه Keras:** یک کتابخانه رایگان متن‌باز با کاربرد آسان برای توسعه و ارزیابی مدل‌های یادگیری عمیق است. این کتابخانه به ما این امکان را میدهد که فقط در چند خط کد مدل‌های شبکه عصبی را پیاده‌سازی کنیم. این کتابخانه شامل پیاده‌سازی انواع مختلف اجزای شبکه‌های عصبی از جمله لایه‌های مختلف، واحدهای مختلف، معماری‌های مختلف و الگوریتم‌های بهینه‌سازی مختلف میباشد که کار با آنها بسیار ساده است.
- در فصل‌های بعدی به توضیح بخش‌های مختلف پیاده‌سازی شده و توضیح کدهای 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 غیر کلیدی (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 کتابخانه‌های 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. ورودی‌های این تابع بصورت زیر میباشند:

- **num**: یک عدد صحیح که تعداد فایل‌هایی 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 بخش‌ها از نرخ نمونه‌برداری 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 فایل‌های صوتی ضبط شده توسط تابع را نشان میدهد. به صورت پیشفرض مقدار آن برابر با ۱ قرار داده شده است.

خروجی این تابع ذخیره فایل‌های صوتی ضبط شده به تعداد آورده شده در ورودی آن میباشد که این فایل‌ها در همان فولدري که کد در آن قرار دارد ذخیره میشوند.

نحوه نامگذاری فایل‌های 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 فایل‌های موجود در فولدر کد بررسی میشود تا اگر از اجراهای قبلی فایلی با فرمت نام‌گذاری این برنامه وجود دارد، شماره‌ی فایل‌ها برای این اجرا از یک شماره بعد از فایل‌های 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 اندازه‌ی seconds ثانیه میکند. برای ضبط صدا از طریق میکروفون از تابع rec از کتابخانه‌ی sounddevice استفاده شده است. این تابع به عنوان ورودی تعداد نمونه‌های مورد نیاز برای ضبط (ورودی اول) که در این تابع این مقدار با ضرب نرخ نمونه‌برداری در طول هر فایل صوتی به ثانیه بدست می‌آید. ورودی دوم این تابع نرخ نمونه‌برداری را مشخص میکند و ورودی سوم هم تعداد کانال‌ها برای ضبط صدا از طریق میکروفون را مشخص میکند. در ادامه با استفاده از تابع write از کتابخانه‌ی scipy سیگنال صوتی ضبط شده از طریق میکروفون را در یک فایل با فرمت wav ذخیره میکند.

۴-۲ بررسی کلی داده‌ها

برای بررسی کلی داده‌ها و اطلاع از پر تعدادترین کلمات بازگو شده در فایل‌های صوتی دیتاست و جداسازی فایل‌هایی که حاوی کلمات کلیدی هستند و انتقال آنها به فولدری مجزا کد data_prep.py پیاده‌سازی 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 کتابخانه‌های مورد استفاده در این کد را به آن اضافه میکنیم. از کتابخانه‌ی **pandas** برای خواندن و کار با فایل محتوای صوت‌های موجود در دیتاست و کار با دیتا در فرمت جدول استفاده میشود. از کتابخانه‌ی **shutil** برای انتقال فایل‌ها به فولدرهای مجزا استفاده شده است و از کتابخانه‌ی **librosa** برای کار با فایل‌های صوتی استفاده شده است. کتابخانه‌ی **soundfile** نیز برای تبدیل و ذخیره سیگنال‌های 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 فایل‌های صوتی که در فرمت **tsv** یا **tab separated values** میباشد را با استفاده از کتابخانه‌ی **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 بعدتر برای شمارش تعداد تکرار هر کلمه مورد استفاده قرار میگیرد. برای اینکار یک حلقه بر روی تمامی سطرهای جدول **df** میزنیم و برای هر سطر، ستون **sentence** که جمله‌ای که در هر فایل صوتی گفته‌شده را مشخص میکند، این جمله را با جداسازی کلماتش در متغیر **a** ذخیره میکنیم و هر کلمه‌اش 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 کلمه‌ی “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 فایل‌هایی 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 کلمه‌ی 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، فایل‌هایی که حاوی کلمه‌کلیدی هستند را جدا کنیم.

```
#----- move validated datas to a sepearte 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 (جمله‌ی 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 فایل‌های صوتی این دیتاست پس از دانلود در یک فولدر می‌باشند و برای جداسازی فایل‌های 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 فایل‌های داخل آن فولدر را بازمیگرداند، استفاده شده است.

```
# sepearte 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 فایل‌هایی که کلمه کلیدی در آنها گفته شده است بود، این فایل‌ها را به یک فولدر جداگانه انتقال می‌دهیم. برای اینکار از تابع **move** از کتابخانه‌ی **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` پیاده‌سازی شده است که این عملیات به فرم تابع در این کد پیاده‌سازی 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 فایل‌هایی که حاوی کلمات کلیدی هستند را در یک فایل CSV ذخیره می‌کند عمل می‌کند. این تابع دو ورودی دارد که ورودی اول با نام `path` می‌باشد که آدرس فایل رونوشت‌های فایل‌های 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 سطرهای 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` میباید 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 `shutil` 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 کتابخانه‌ی **librosa** برای خواندن فایل‌های 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 فایل‌های صوتی به قسمت‌های ۰٫۶ ثانیه‌ای می‌باشد. دلیل اینکار بجای تقسیم فایل‌ها به تکه‌های ۱ ثانیه‌ای این بود که در قسمت‌های یک ثانیه‌ای معمولاً بیش از یک کلمه گفته می‌شد و برای اینکه تکه‌ها حتماً حاوی تنها یک کلمه باشند، طول آنها را به ۰٫۶ تغییر دادیم. هر یک از این قسمت‌ها هم با گام‌هایی به طول ۰٫۵ ثانیه در طول سیگنال اصلی حرکت میکنند تا تکه‌ها با هم اشتراک داشته باشند و کلمه‌ای در وسط بازگو شدن آن قطع نشود و از دست نرود.

ورودی‌های این تابع به صورت زیر می‌باشند:

- **path**: آدرس فولدری که شامل فایل‌های صوتی می‌باشد که ما می‌خواهیم آنها را به تکه‌های ۰٫۶ ثانیه‌ای تقسیم کنیم.
 - **sr**: نرخ نمونه برداری که با استفاده از آن تعداد نمونه‌ها در ۰٫۶ ثانیه یا ۰٫۵ ثانیه محاسبه می‌شود.
- خروجی این تابع، دخیره سازی تکه‌های 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

```

نحوه‌ی کار این تابع را در تصویر بالا مشاهده میکنید. ابتدا لیست فایل‌های موجود در فولدر در متغیر `li` ذخیره میشود. سپس حلقه‌ای بر روی این لیست تعریف میشود که ابتدا هر فایل توسط تابع `load` از کتابخانه‌ی `librosa` خوانده میشود و این سیگنال به تکه‌هایی به طول 0.6 ($0.6 = 22050 * 0.6$) و با گام‌های 0.5 ثانیه‌ای ($0.5 = 22050 * 0.5$) تقسیم شده و این تکه‌ها با استفاده از تابع `write` از کتابخانه‌ی `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 فایل‌های 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 فایل‌های داخل فولدر خوانده میشود. سپس لیستی با نام `trues` ایجاد میشود که در نهایت اسامی فایل‌هایی که حاوی کلمه کلیدی میباشند را در خود نگه میدارد. در قسمت بعدی یک حلقه بر روی فایل‌ها زده میشود که در هر تکرار برای هر فایل، ابتدا فایل خوانده میشود و سپس توسط تابع `play_buffer` از کتابخانه‌ی `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 کتابخانه‌ی `pandas` تبدیل میشود و سپس در فرمت `CSV` ذخیره میشود.

۴-۴ ویژگی‌های قابل استخراج و استفاده در سیگنال‌های صوتی

در این قسمت به انواع ویژگی‌های قابل استخراج از سیگنال‌های صوتی که در مسائل یادگیری ماشین کاربرد دارند میپردازیم. برای یادگیری الگوها با استفاده از مدل‌های یادگیری ماشین و یادگیری عمیق به داده‌هایی از جنس عددی نیاز داریم. سیگنال صوتی به صورت یک موج مکانیکی و پیوسته در زمان میباشد. در کامپیوترها برای پردازش صوت سیگنال پیوسته زمان آن را با استفاده از نمونه‌برداری به سیگنال گسسته تبدیل میکنند. پیش تر به نرخ نمونه‌برداری اشاره شد که این نرخ تعداد نمونه‌ها در یک ثانیه سیگنال را مشخص میکنند. در ادامه به ویژگی‌های قابل استخراج از سیگنال صوتی میپردازیم.

۴-۴-۱ سیگنال صوتی خام

فایل‌های صوتی در کامپیوترها به شکل دنباله‌ای از اعداد ذخیره میشوند که این اعداد نشان دهنده‌ی دامنه‌ی موج در هر نمونه میباشند. این ویژگی (دامنه موج) در حوزه‌ی زمانی میباشد و اطلاعاتی در رابطه با تغییرات دامنه در طول زمان به ما میدهد. از این

سیگنال میتوان به عنوان ورودی مدل‌های 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 `librosa` میتوان سیگنال دامنه/زمان را از فایل‌های صوتی استخراج کرد. این تابع آدرس فایل را بعنوان ورودی دریافت میکند و سیگنال و نرخ نمونه‌برداری را بعنوان خروجی بازمیگرداند.

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

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

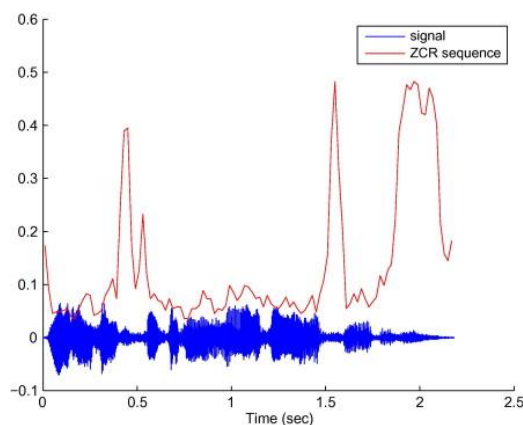
فریم بندی سیگنال:

در این پروسه سیگنال به قسمت‌هایی با طول مساوی تقسیم میشود. به هریک از این قسمت‌ها فریم گفته میشود. طول هر فریم معمولاً بین ۲۵۶ تا ۸۱۹۲ میباشد.

۴-۴-۲ zero-crossing rate

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 بخش‌هایی از صوت که مکالمه در آن صورت گرفته مورد استفاده قرار میگیرد چراکه دامنه در صوتی که مکالمه‌ی انسان در آن وجود دارد بر خلاف صدای محیط، به دفعات زیاد بین مقادیر مثبت و منفی تغییر میکند.



تصویر شماره ۳۲

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.

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

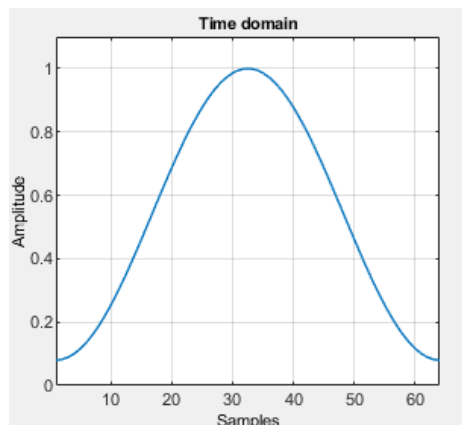
تصویر شماره ۳۳

از دیگر دسته ویژگی‌ها که قابل استخراج از سیگنال‌های صوتی میباشد میتوان به ویژگی‌ها در حوزه فرکانسی اشاره کرد. برای رفتن از یک سیگنال در حوزه زمان به حوزه فرکانس باید ابتدا همانند قبل سیگنال را به فریم‌ها 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 مولفه‌هایی 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:

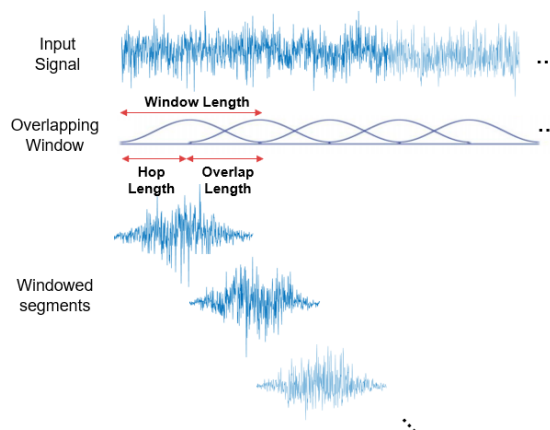
$$w(k) = 0.5 - 0.5 \cos\left(\frac{2\pi k}{N-1}\right)$$

تصویر شماره ۳۴



تصویر شماره ۳۵

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 می‌آید. 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 فریم‌هایی 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 گام‌های جلو رفتن فریم‌ها را کوچکتر از طول هر فریم در نظر بگیریم. به طول این گام‌ها hop length گفته میشود و با استفاده از این روش فریم‌ها با هم اشتراک دارند و دیگر اطلاعاتی از دست نمی‌رود. در تصویر زیر میتوان این فرایند را مشاهده کرد.



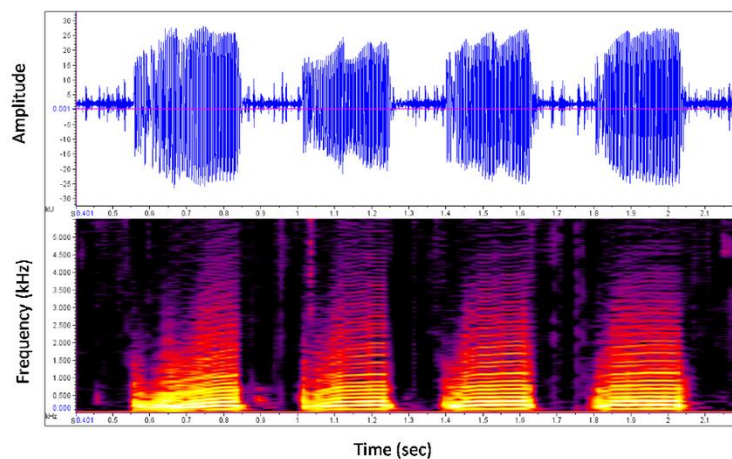
تصویر شماره ۳۶

پس از فریم بندی سیگنال با گام‌های کوچکتر از طول فریم و اعمال تابع windowing روی فریم‌ها، 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.

۴-۴-۴ Spectrogram

اسپکتروگرام (spectrogram) یک نمایشی از طیف فرکانسی سیگنال در طول زمان است. اسپکتروگرام یک نقشه رنگی هست که برحسب زمان و فرکانس نمایش داده می‌شود 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 می‌آید. 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 خروجی‌ها را به همان ترتیب فریم‌ها کنار هم بچینیم، یک ماتریس از ضرایب بدست می‌آید 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 شبکه‌های عصبی کانولوشنال 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:



تصویر شماره ۳۷

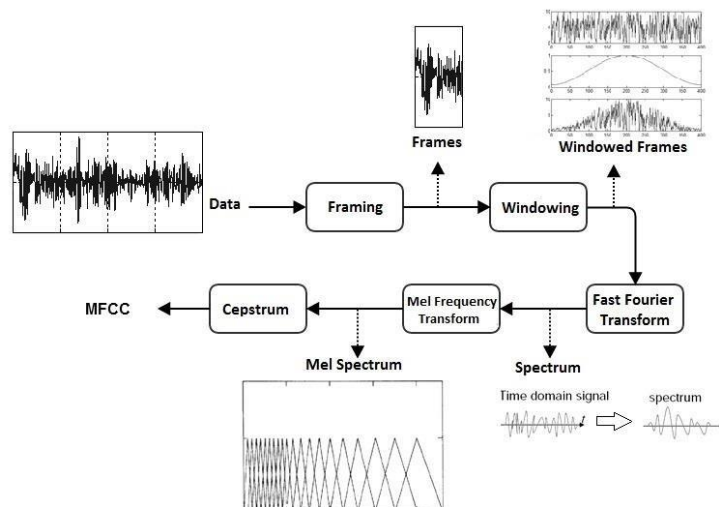
۵-۴-۴ Mel-Frequency Cepstral Coefficients

متداول ترین و کارا ترین ویژگی‌ها برای بازشناسی گفتار، Mel-Frequency Cepstral Coefficients یا به اختصار MFCC ها هستند. ایده اصلی این ویژگی از نحوه‌ی 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:

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

تصویر شماره ۳۸

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. معمولاً ۹ تا ۱۳ ضریب اول از این تبدیل بیشتر اطلاعات سیگنال صوتی را در خود نگهداری میکنند و در مسائل یادگیری ماشین معمولاً از ۱۳ سیگنال اول استفاده میشود. مراحل محاسبه این ضرایب به صورت زیر میباشد:



تصویر شماره ۳۹

در تصویر بالا تمامی مراحل تبدیل سیگنال خام در حوزه زمان تا تبدیل آن به MFCC آورده شده است. مرحله اول تبدیل سیگنال به فریم‌ها 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 فرکانس‌های 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. معمولاً 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 مدل‌ها می‌باشد، می‌پردازیم. ابتدا کتابخانه‌های استفاده شده در این کد را اضافه میکنیم.

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

کتابخانه‌ی librosa برای خواندن فایل‌های صوتی، کتابخانه‌ی json برای ذخیره سازی دیتاست نهایی در فرمت json و کتابخانه‌ی 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 پیاده‌سازی 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 قسمت‌هایی به طول ۱ ثانیه و با گام‌هایی به طول 0,5 ثانیه می‌باشد. دلیل استفاده از این تابع این است که فایل‌های کلاس `false` (فایل‌هایی که کلمه‌کلیدی در آنها گفته نشده است) طولی بیشتر از ۱ ثانیه دارند و چون ورودی مدل‌های کانوولوشنال باید ابعاد یکسانی داشته باشند، باید به قطعات ۱ ثانیه‌ای تبدیل شوند. این تابع سیگنال اصلی را به عنوان ورودی دریافت می‌کند و آنرا به قسمت‌های یک ثانیه‌ای تبدیل می‌کند و این قسمت‌ها 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 فولدري که شامل فولدر کلاس‌های نهایی می‌باشد را نشان می‌دهد. در این فولدر به تعداد کلاس‌های نهایی فولدر وجود دارد که هم نام کلاس‌ها می‌باشند. در این فولدرها فایل‌های مربوط به هر کلاس آورده شده است. در مسئله ما این کلاس‌ها `true` (داده‌هایی که کلمه‌کلیدی در آن گفته شده) و `false` (داده‌هایی که کلمه‌کلیدی 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. کلیدهای این دیکشنری به صورت زیر می‌باشند:

- `map`: مقادیر این کلید لیستی از کلاس‌های موجود در دیتاست را نشان می‌دهد.
- `labels`: لیستی از برچسب‌های هر داده موجود در دیتاست.
- `MFCC`: لیستی از ویژگی‌های استخراج شده برای هر داده.
- `File`: لیستی از نام فایل‌های داده‌های موجود در دیتاست.

تمامی این لیست‌ها 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 ویژگی‌های 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 فولدرها یک حلقه ایجاد میکنیم. نام فولدرها همانطور که در قسمت قبل گفته شد در لیست map آورده شده است. در این حلقه ابتدا لیست فایل‌های داخل این فولدر در متغیر files ریخته میشود و ترتیب این لیست توسط تابع shuffle از کتابخانهی random به یک ترتیب تصادفی تبدیل میشود چراکه در ابتدا بر اساس الفبا مرتب شده اند و فایل‌هایی با گوینده یکسان پشت سر هم قرار گرفته اند. سپس یک حلقه روی اسامی فایل‌ها زده میشود تا ویژگی‌ها 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 قسمت‌های ۱ ثانیه‌ای تقسیم کنیم. از ابتدای حلقه‌ی اول یک متغیر با نام count که مقدار اولیه آن ۰ میباشد تعریف کردیم تا نسبت کلاس‌ها 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 داده‌ها با کلاس true بسیار کمتر از کلاس false میباشد، نیاز است که نسبت تعداد داده‌های هر کلاس در دیتاست کنترل شود. اگر مقدار متغیر count برای مثال از ۶۰۰۰ بیشتر شد، دیگر به داده‌های این کلاس اضافه نمیکنیم. سپس برای هر قسمت یک ثانیه‌ای، ضرایب mfcc را استخراج میکنیم. برای اینکار از تابع mfcc از کتابخانهی librosa استفاده میکنیم. ورودی‌های 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 نمونه‌برداری 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 ورودی‌ها 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 (22050 / 512) ۴۴ 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 فایل‌ها 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 فایل `data.json` ذخیره میشود.

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

۴-۶ راه حل های موجود برای دیتاست نامتوازن

مشکل نامتوازن بودن کلاس‌های داده‌ها در دیتاست برای مسائل دسته‌بندی میتواند در عملکرد مدل‌ها تاثیر منفی بگذارد. برای مثال اگر کلاس `true` ۰,۱ داده‌ها را تشکیل بدهد، مدل میتواند همیشه کلاس `false` را پیشبینی کند و دقت آن هم ۰,۹ بشود در حالی که برای تمامی اعضای کلاس `true` اشتباه پیشبینی کرده است. برای حل این مشکل راه‌های مختلفی وجود دارد که در این قسمت به آن‌ها میپردازیم. یکی از راه حل‌های 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 نمونه‌ها در کلاسی که تعداد کمتری دارد گفته میشود. راه‌های زیادی برای اینکار وجود دارد که یکی از این‌ها تکرار داده‌ها در کلاس کمتر میباشد. به این معنا که هر نمونه عضو این کلاس را تا جایی که نسبت کلاس‌ها 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 داده‌های فعلی میباشد. برای اینکار میتوانیم با اعمال تغییرات روی داده‌های فعلی، داده‌های جدیدی را تولید کنیم و تعداد داده‌ها در این کلاس را افزایش دهیم. برای مثال در داده‌های تصویری میتوان با چرخش یا حذف بخشی از تصویر یا اضافه کردن نویز به تصویر به این مهم دست یافت. در داده‌های صوتی نیز میتوان با تغییر `pitch`، اضافه کردن نویز، تغییر سرعت داده‌های جدیدی تولید کرد. در ادامه کدی که برای پیاده سازی این روش نوشته شد را بررسی میکنیم.

برای استفاده از `upsampling` بر روی داده‌های موجود در دیتاست، کد `upsample.py` پیاده‌سازی شده است.

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

ابتدا کتابخانه‌های مورد استفاده در این کد را اضافه میکنیم. تمامی این کتابخانه‌ها در بخش‌های 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 noise 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 پیاده‌سازی 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 `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 سیگنال‌هایی با طول کمتر از ۱ ثانیه، با اضافه کردن دامنه صفر به ابتدا و انتهای آنها به صورت رندوم، به سیگنال‌های ۱ ثانیه‌ای تبدیل میشوند. ورودی اول این تابع آدرس فولدری است که فایل‌های صوتی در آن قرار دارند. ورودی دوم نرخ نمونه برداری این فایل‌ها را مشخص میکند. هدف پیاده‌سازی این تابع تصحیح طول سیگنال‌هایی بود که در قسمت‌های 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 فایل‌های 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 نمونه‌هایی که باید سمت چپ سیگنال (ابتدای سیگنال) مشخص میشود و به همین تعداد نمونه‌هایی با دامنه ۰ به سمت چپ سیگنال اضافه میکنیم. سپس نمونه‌های 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 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 داده‌های کلاس کمتر به روش `upsampling` پیاده‌سازی شده است. ورودی این تابع آدرس فولدری است که شامل فایل‌های این کلاس بوده و خروجی آن ذخیره فایل‌های 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 فایل‌های 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 سیگنال‌ها به همراه دو کپی از سیگنال اصلی در فولدر `augmented` ذخیره می‌شوند.

۴-۶-۲ downsampling

روش دیگر برای حل مشکل نامتوازن بودن دیتاست، `downsampling` نام دارد. در این روش تعداد نمونه‌ها در کلاسی که تعداد بیشتری دارد را با حذف نمونه‌ها از آن کم می‌کنیم. این روش در پروژه ما قابل استفاده نبود چراکه شبکه‌های عصبی برای یادگیری نیاز به مقادیر زیادی داده دارند و این روش تعداد داده‌ها 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 مدل‌ها میپردازیم. برای شروع کار از یک معماری معمول و متداول برای شبکه‌ها استفاده کردیم که جلوتر به جزئیات آن اشاره میشود. سپس به بررسی عملکرد مدل‌ها میپردازیم و در نهایت راه حل‌های استفاده شده برای بهبود عملکرد مدل را بررسی میکنیم.

۵-۱-۱ CNN مدل

در این قسمت به جزئیات مدل شبکه کانولوشنال 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 (Batch Normalization)	(None, 42, 11, 64)	256
max_pooling2d_30 (MaxPooling2D)	(None, 21, 6, 64)	0
conv2d_31 (Conv2D)	(None, 19, 4, 32)	18464
batch_normalization_31 (Batch Normalization)	(None, 19, 4, 32)	128
max_pooling2d_31 (MaxPooling2D)	(None, 10, 2, 32)	0
conv2d_32 (Conv2D)	(None, 9, 1, 32)	4128
batch_normalization_32 (Batch Normalization)	(None, 9, 1, 32)	128
max_pooling2d_32 (MaxPooling2D)	(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: 34,690		
Trainable params: 34,442		
Non-trainable params: 256		

تصویر شماره ۴۰

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

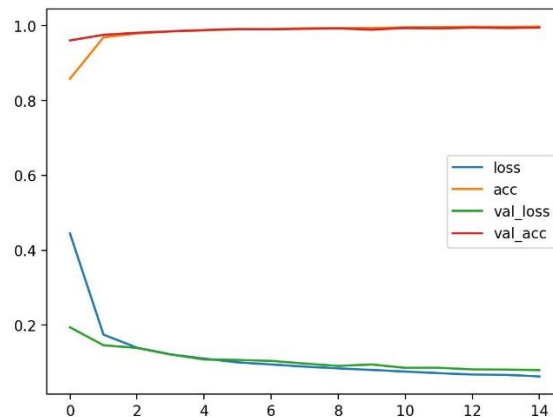
- لایه ورودی که یک لایه کانولوشنال با ۶۴ فیلتر ۳ در ۳ میباشد.
- لایه بعدی که یک لایه batch normalization است. این لایه برای نرمال کردن خروجی‌های لایه قبلی میباشد.
- لایه max_pooling با فیلتر ۳ در ۳ و گام‌های (2x2)
- سه لایه بعدی تکرار لایه‌های بالا با این تفاوت که در لایه کانولوشنال دارای ۳۲ فیلتر میباشد.
- سه لایه بعدی هم مانند سه لایه قبلی هستند. با این تفاوت که فیلترهای max_pooling آن ۲ در ۲ میباشد.
- لایه بعدی لایه‌ی flatten میباشد که ماتریس را به یک آرایه تک بعدی تبدیل میکند.
- لایه بعدی یک لایه‌ی fully connected با ۶۴ نرون و تابع فعالسازی relu میباشد.
- لایه بعدی یک لایه‌ی dropout با ۶۴ نرون و ضریب ۰,۳ میباشد. کاربرد این لایه غیر فعالسازی نرون‌ها 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 لایه‌ی خروجی هست (۲ کلاس خروجی داریم). تابع فعالسازی این لایه softmax میباشد چراکه به یک احتمال برای هر کلاس نیاز داریم.

در تصویر زیر میتوان مقادیر اندازه گیری عملکرد مدل که در بخش ۳-۳ توضیح داده شد را مشاهده کرد:

	precision	recall	f1-score	support
0	0.99	1.00	0.99	1802
1	0.95	0.93	0.94	203

تصویر شماره ۱

همچنین مقدار خطا و دقت روی مجموعه دادگان آموزش را در تصویر زیر مشاهده میکنید.



تصویر شماره ۲

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.

۲-۵ 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 کتابخانه‌ی `sklearn` برای تقسیم دیتاست به دو بخش مجموعه دادگان آموزش و تست استفاده شده است. باقی کتابخانه‌ها در قسمت‌های 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 آماده‌سازی دیتاست برای استفاده از آنها توسط مدل، پیاده‌سازی 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. مدل‌ها به دو دسته دیتا نیاز دارند: ورودی‌ها که ویژگی‌های استخراج شده از سیگنال‌های صوتی هستند (کلید `MFCC` در دیتاست) و خروجی‌ها 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 داده‌ها بعنوان مجموعه تست و باقی داده‌ها به عنوان مجموعه آموزش (چراکه برای آموزش به داده‌های 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 مجموعه‌ها 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,# coeffs])
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 داده‌ها 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]

داده‌های فعلی ما تنها حاوی دو بعد اول می‌باشند که تعداد قسمت‌های هر سیگنال و تعداد ضرایب استخراج شده را نشان می‌دهد. بعد آخر تعداد کانال‌های 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 (Batch Normalization)	(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 Normalization)	(None, 20, 5, 16)	64
max_pooling2d_1 (MaxPooling2D)	(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		

تصویر شماره ۴۳

مدل پیاده‌سازی شده در این کد یک نمونه ساده‌تر از مدل توضیح داده شده در بخش قبلی می‌باشد. دلیل این ساده‌سازی مشکل **overfitting** بود که در بخش بعدی به آن خواهیم پرداخت. برای مثال تعداد لایه‌های کانولوشن یکی کمتر از مدل قبلی می‌باشد و تعداد فیلترهای آن هم کمتر از مدل قبلی است. در ادامه به پیاده‌سازی 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 دیتاست‌ها بدست آورده در متغیر `input_shape` ذخیره می‌کنیم. سپس فاکتور نرخ یادگیری را برابر با عدد 0.0001 قرار می‌دهیم. سپس مدل را توسط تابع `Sequential` از کتابخانه `keras` ایجاد می‌کنیم. این مدل به صورتی می‌باشد که می‌توان به آن با استفاده از توابع موجود در کتابخانه `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 کانولوشن 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 0.005 (این `L2_regularizer` باعث اضافه کردن یک پنالتی برای اندازه وزن‌ها به تابع خطا می‌شود که از بزرگ شدن وزن‌ها 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 لایه‌های 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 لایه‌های fully connected به شبکه اضافه میشوند. این لایه‌ها 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 فعالسازى 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 بهینه‌ساز 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 داده‌ها (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 کانولوشنال:

```
test error: 0.03897948808447827
test accuracy: 0.9945137
```

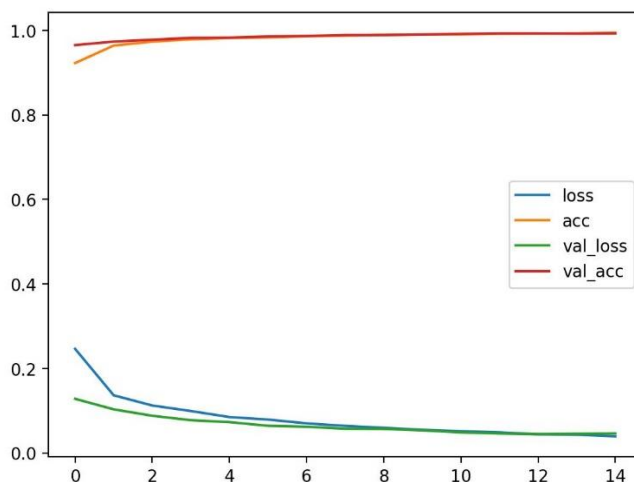
تصویر شماره ۴۴

در تصویر زیر نیز عملکرد مدل را مشاهده میکنید:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1803
1	0.98	0.96	0.97	202

تصویر شماره ۴۵

همانطور که در تصویر بالا مشاهده میشود، دقت مدل برابر با عدد ۰٫۹۸ میباشد که این عدد به علت نوع مسئله میباشد و این مدل از مشکلات `overfitting` به اندازه مدل قبلی رنج نمیبرد و برای داده‌های جدید به خوبی عمل میکند.



تصویر شماره ۴۶

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

برای پیشبینی خروجی توسط مدل‌ها کد `predict.py` پیاده‌سازی 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` پیاده‌سازی 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 `librosa` انجام میدهیم.

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

سپس ابعاد ورودی را برای استفاده 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 ویژگی‌های 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 حل‌های استفاده شده برای overfitting

همانطور که پیشتر اشاره شد مدل اول که پیاده‌سازی شد مشکل overfitting داشت به این معنا که تنها داده‌های 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 پیاده‌سازی شد که در قسمت‌های قبلی توضیح داده شد. از دیگر اقدامات میتوان به ساده کردن معماری مدل اشاره کرد که در قسمت پیاده‌سازی مدل توضیح داده شد. برای اینکار تعداد لایه‌های 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 نرون‌ها در هر epoch میشود که این امر از تغییر وزن‌های این نرون‌ها جلوگیری میکند و از حفظ کردن داده‌ها توسط مدل و overfit شدن جلوگیری میکند. از دیگر اقدامات میتوان به کم کردن تعداد epochها اشاره کرد. این امر باعث کمتر دیده شدن داده‌ها 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 صوت‌هایی با محتوای صدای محیط و نویز به صورت دستی ضبط و به مجموعه دادگان اضافه شدند.

۵-۴ مقایسه عملکرد مدل‌ها

در این قسمت به مقایسه عملکرد مدل‌های پیاده‌سازی شده در این پروژه میپردازیم. در جدول زیر میتوان معیارهای ارزیابی مختلف را برای هر کدام از این مدل‌ها مشاهده کرد.

مدل	خطای تست	دقت تست	کلاس	precision	Recall	F1-score
شبکه عصبی کاتولوشنال اولیه	0.043	0.98	کلمه کلیدی (کلاس 1)	0.95	0.93	0.94
			کلاس 0	0.99	1.00	0.99
شبکه عصبی کاتولوشنال بهبود یافته	0.038	0.99	کلمه کلیدی (کلاس 1)	0.98	0.96	0.97
			کلاس 0	1.00	1.00	1.00
شبکه عصبی یازگشتی	0.076	0.97	کلمه کلیدی (کلاس 1)	0.89	0.84	0.87
			کلاس 0	0.98	0.99	0.99

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.

کارهای انجام شده در این بخش به همان ترتیبی که فصول این گزارش آورده شده است بودند که شامل جمع‌آوری داده‌ها، آماده‌سازی داده‌ها برای مدل‌های یادگیری عمیق و در نهایت انتخاب جزئیات شبکه‌های عصبی و پیاده‌سازی آنها برای استفاده در برنامه بود.

۶-۲ نتیجه گیری

امروزه پردازش صوت در محصولات شرکت‌های مختلف نقش مهمی را بازی میکند و در بسیاری موارد میتواند کاربرد داشته باشد. با پیشرفت‌های صورت گرفته در حوزه هوش مصنوعی و یادگیری عمیق و با جمع‌آوری داده‌ها به صورت هدفمند، عملکرد مدل‌ها بسیار بهبود یافته و در مواردی از انسان هم بهتر عمل کرده است. با استفاده از متد‌های یادگیری عمیق و در دسترس داشتن مقدار داده به اندازه کافی میتوان به مدل‌هایی با عملکرد بسیار خوب دست یافت که در این پروژه سعی شد تا یک نمونه از کاربردهای پردازش صوت پیاده‌سازی شود و به عملکرد مطلوبی هم دست یافتیم.

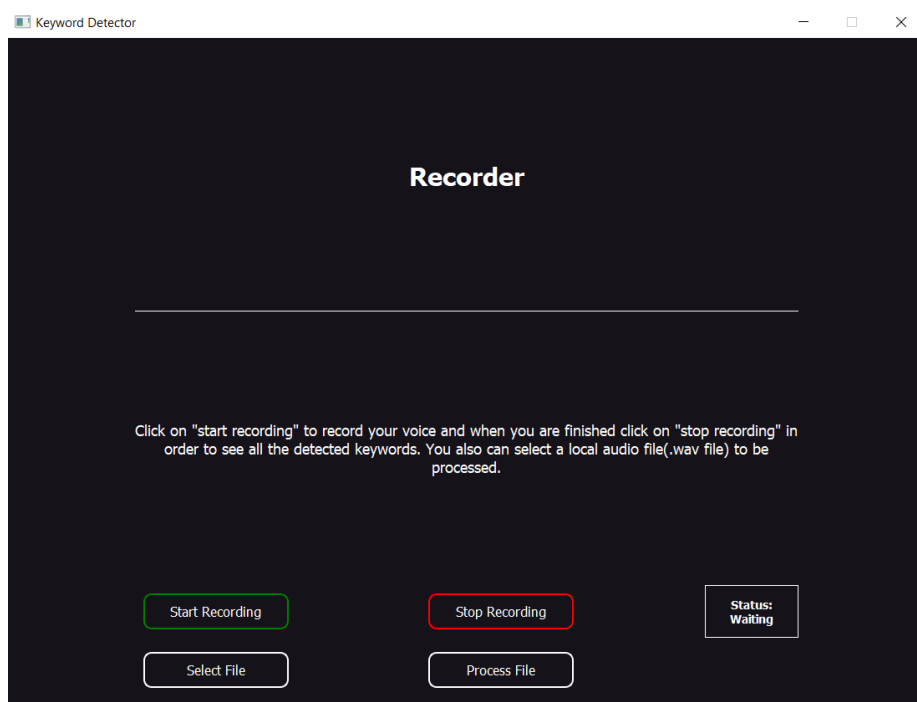
مراجع

- [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 kavukeoglu. 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

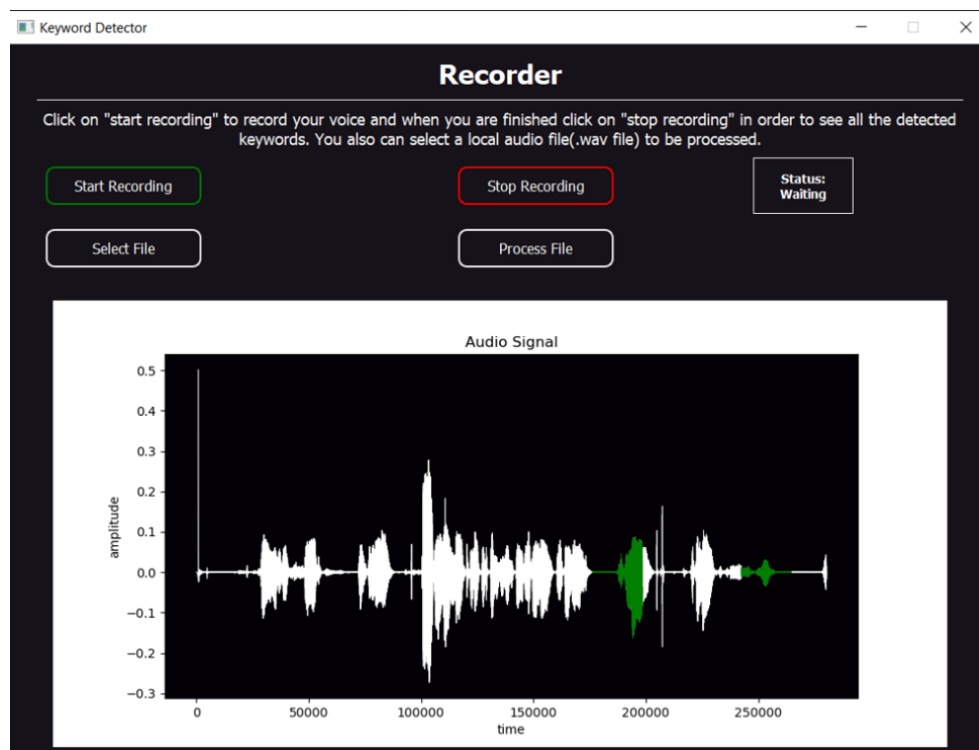
نحوه کار با برنامه دسکتاپ

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



تصویر شماره ۴۷

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:

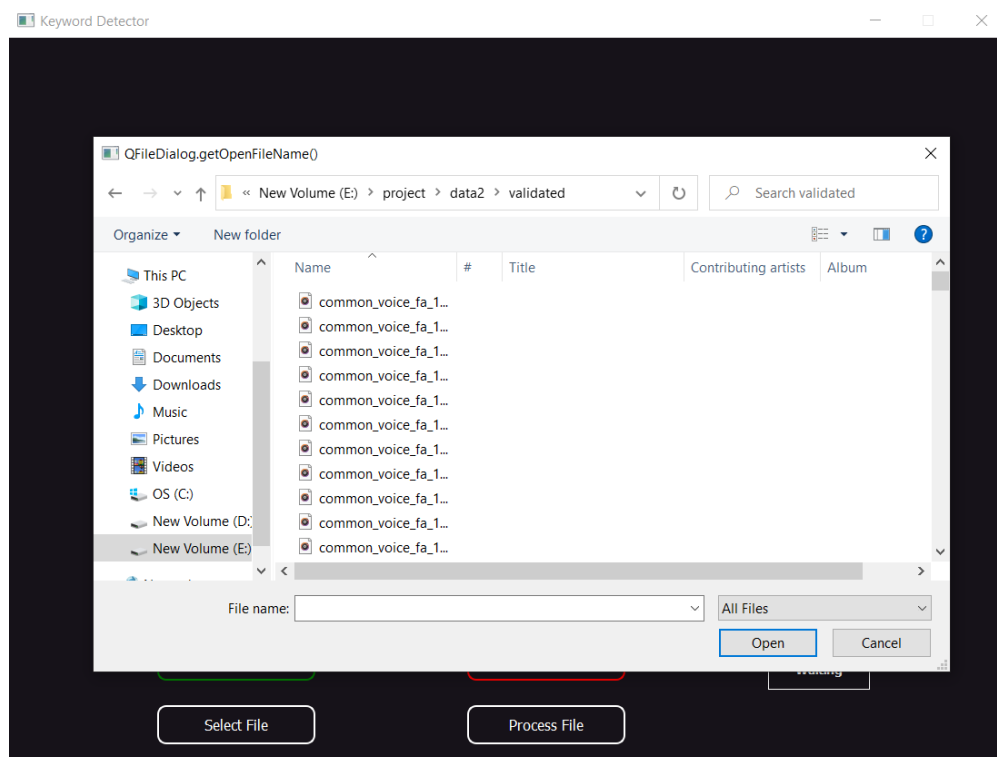


تصویر شماره ۴/۸

که در آن سیگنال صوت ضبط شده نمایش داده میشود و قسمت‌های سبز نشان دهنده‌ی بخش‌هایی از صوت است که برنامه کلمه کلیدی را تشخیص داده است. همچنین در فولدری با نام detected فایل‌های صوتی ۱ ثانیه‌ای از بخش‌هایی 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:



تصویر شماره ۴۹



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