

دانشگاه صنعتي امیرکبیر  
(پلی تکنیک تهران)

دانشكده مهندسی برق

ﮔﺰﺍﺭﺵکار پروژه نهایی

شبکه های چند رسانه­ای

Dodge Lane

نگارش

علی بهمنیار

پارسا پیله­ور

علی بابالو

پویا شریفی

استاد راهنما

دکتر شریفیان

تیر ماه ۱۴۰۲

تشخیص گفتار یا به اصطلاح speech recognition همواره از موضوعات مورد توجه تحقیقات هوش مصنوعی بوده است، در این پروژه به پیاده‌سازی یک سامانه‌ی تشخیص گفتار برای تشخیص فرامین صوتی و سپس پیاده‌سازی  یک بازی دو بعدی توسط کتابخانه‌ی PyGame برای بهره‌گیری از این سامانه پرداخته شده است.

ما توانستیم با دو روش مدل هایی طراحی کنیم و بازی ایجاد شده را با کمک صوت کنترل کنیم.مدل های ما ۹۴.۳۷ درصد accuracy و ۹۲.۶۲ درصد accuracy توانستیم به نتایج مطلوب برسیم.

\*\* اگر نیاز به train کردن شبکه دارید، فایل های دیتاست، ضرایب MFCC و Spectrogram بعلت حجیم بودن در کورسز آپلود نکردیم اما در لینک درایو زیر در دسترس است:

https://drive.google.com/file/d/1xoYZsE4EUadnmuRhex7ILZwG6v1SRtYR/view?usp=sharing

فهرست مطالب

**پیش گزارش** ...............................................................................................................................................................۱

ساختار شبکه CNN .....................................................................................................................................۱

لایه کانوولوشنی .............................................................................................................................................۲

لایه ادغام .........................................................................................................................................................۲

لایه کاملا متصل ............................................................................................................................................۳

**شرح دیتاست** ..............................................................................................................................................................۳

دیتاست ............................................................................................................................................................۳

نحوه استخراج ویژگی ها ..............................................................................................................................۳

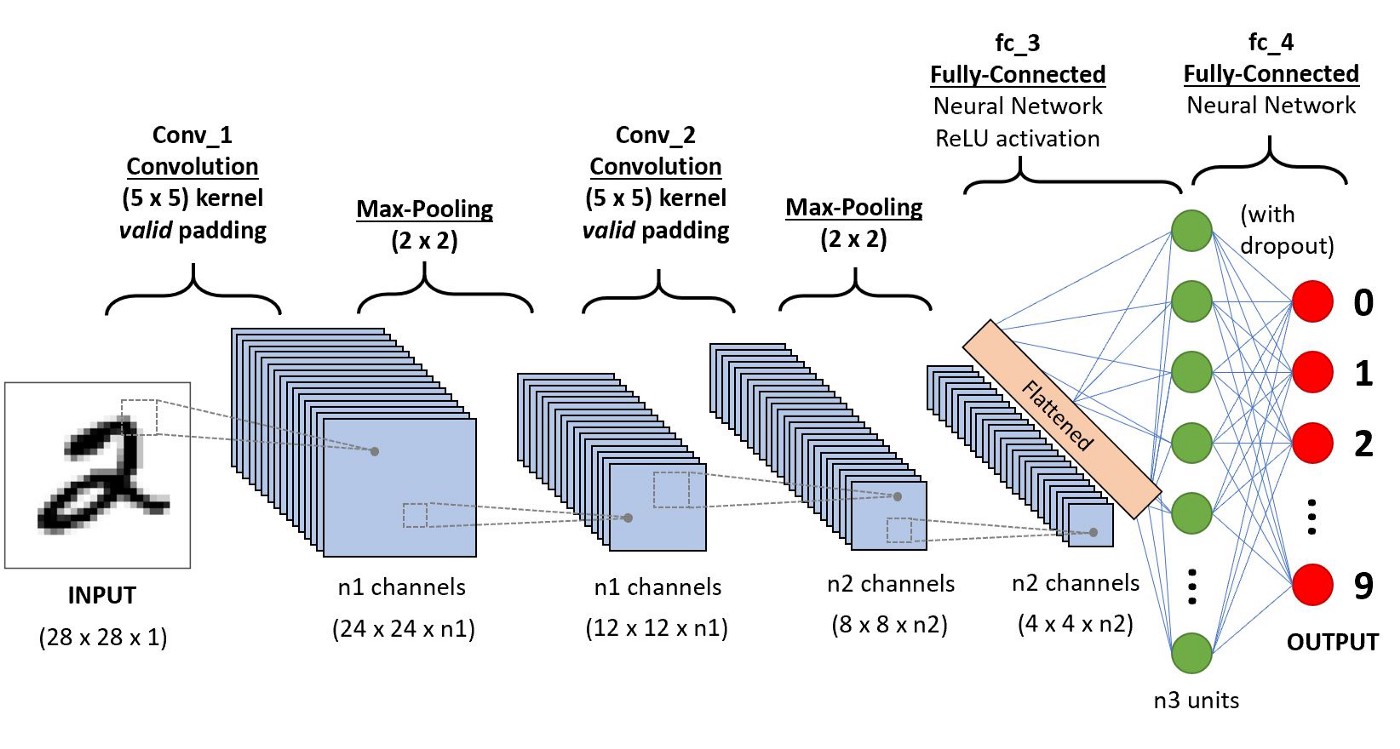
**استفاده از ضرایب MFCC در شبکه MLP** ................................................................................................۴

**استفاده از spectrogram به عنوان تصویر و استفاده از شبکه‎های CNN** .................................۱۲

**Dodge Lane** ........................................................................................................................................................۱۷

# پیش گزارش:

**ساختار شبکه CNN :**

شبكه كانولوشني از لايه هاي اصلي زير تشكيل شده است:

شکل 1) نمونه ای شبکه عصبی کانوولوشنی

• لایه کانولوشنی

• لایه ادغام

• لایه کاملا متصل

### لايه كانولوشني:

شکل 2) عملگر کانوولوشن

در اين لايه عمل كانولوشن بر روي داده ورودي و با استفاده از تعدادي فيلتر انجام مي شود. براي محاسبه هر درايه خروجي، ماتريس فيلتر بر روي ماتريس ورودي لغزانده مي شود. عمل كانولوشن به اين صورت تعريف مي شود كه ابتدا اولين عنصر فيلتر بر روي اولين عنصر ماتريس ورودي قرار مي گيرد. سپس مجموع ضرب درايه هاي متناظر فيلتر با درايه هاي متناظز ماتريس ورودي محاسبه مي شود. در نهايت فيلتر بر روي تصوير ورودي به اندازه پارامتر از پيش تعيين شده s يه جلو برده مي شود. با تكرار اين مراحل ماتريس خروجي محاسبه مي شود.

### لايه ادغام:

يكي ديگر از لايه هاي شبكه كانولوشني لايه ادغام است. اين لايه هيچ پارامتر آموزشي ندارد. هدف اين لايه كاهش ابعاد ماتريس ورودي ورودي و همزمان حفظ اطلاعات ارزشمند ورودي است. در اين لايه ابتدا ماتريس با ابعاد از پيش تعيين شده k در نظر گرفته مي شود. اين پارامتر معمولا برابر با ۲ در نظر گرفته مي شود. سپس با لغزاندن فيلتر بر روي ورودي اندازه ماتريس ورودي كاهش مي يابد. يك نمونه از عملگرهايي كه براي نمونه برداري در اين لايه استفاده مي شود، عملگر بيشينه است. در اين حالت ماتريس از پيش تعيين شده بر روي داده ورودي لغزانده مي شود و تنها بيشينه عناصري كه در هر بخش قرار مي گيرند را به عنوان خروجي در نظر مي گيرد.

### لايه كاملا متصل:

در اين لايه يك شبكه عصبي كاملا متصل قرار گرفته است. در اين لايه هدف مرتبط كردن ماتريس نهايي با خروجي هاي نهايي شبكه است. وزن هاي شبكه كاملا متصل از طريق پس انتشار خطا بدست مي آيد.

# شرح دیتاست:

**دیتاست:**

برای این پروژه از دیتاست **Speech Command Dataset**، که توسط TensorFlow و AYI جمع آوری شده است برای آموزش شبکه عصبیمان استفاده می­کنیم. این دیتاست شامل ۶۵۰۰۰ دیتا از ۳۰ کلمه کوتاه است که توسط مشارکت عمومی هزاران نفر بدست آمده است.

این دیتاست شامل دیتاهایی با برچسب هایی شامل Yes, No, Up, Down, Left, Right, On, Off, … می­باشد که برای این پروژه ما از لیبل های Up, Down, Left, Rightاستفاده می­کنیم که هر کلاس بیش از ۲۰۰۰ عضو دارد.

**نحوه استخراج ویژگی ها:**

ابتدا دیکشنری data را به صورت زیر تعریف میکنیم:

    data = {

        "mappings": [],

        "labels": [],

        "MFCCs": [],

        "files": []

    }

لیست mappings به تربیب شامل لیبل‎های down, left, right, up خواهد بود. لیست labels شامل لیبل‎های 0، 1، 2، 3 برای هر داده می‎شود. ضرایب mfcc یا spectrogram در لیست MFCCs یا spectrograms قرار خواهند گرفت، همچنین نام هر فایل در را در لیست files ذخیره خواهیم کرد.

برای استخراج فیچرها از کتابخانه librosa استفاده شده است کد زیر فیچرهای mfcc و یا spectrogram را استخراج می‎کند؛

*# loop though all the sub-dirs*

*for* i, (dirpath, dirnames, filenames) *in* enumerate(os.walk(dataset\_path)):

*# ensure that we're not at root level*

*if* dirpath is not dataset\_path:

*# update mappings*

            category = dirpath.split("/")[-1] *# dataset/down -> ['dataset',*

*'down']*

            data["mappings"].append(category)

            print(f"Processing {category}")

*# loop through all the filenames and extract MFCCs*

*for* f *in* filenames:

*# get file path*

                file\_path = os.path.join(dirpath, f)

*# load audio file*

                signal, sr = librosa.load(file\_path)

*# ensure the audio file is at least 1 sec*

*if* len(signal) >= SAMPLES\_TO\_CONSIDER:

*# enforce 1 sec long signal*

                    signal = signal[:SAMPLES\_TO\_CONSIDER]

*# extract MFCCs*

                    MFCCs = librosa.feature.mfcc(signal, *n\_mfcc*=n\_mfcc,

*hop\_length*=hop\_length, *n\_fft*=n\_fft)

*# extract spectrograms*

*# spectrograms = librosa.stft(signal, n\_fft=n\_fft,*

*hop\_length=hop\_length)*

*# store data*

                    data["labels"].append(i-1)

                    data["MFCCs"].append(MFCCs.T.tolist())

*# data["spectrograms"].append(spectrograms.T.tolist())*

                    data["files"]. append(file\_path)

                    print(f"{file\_path}: {i-1}")

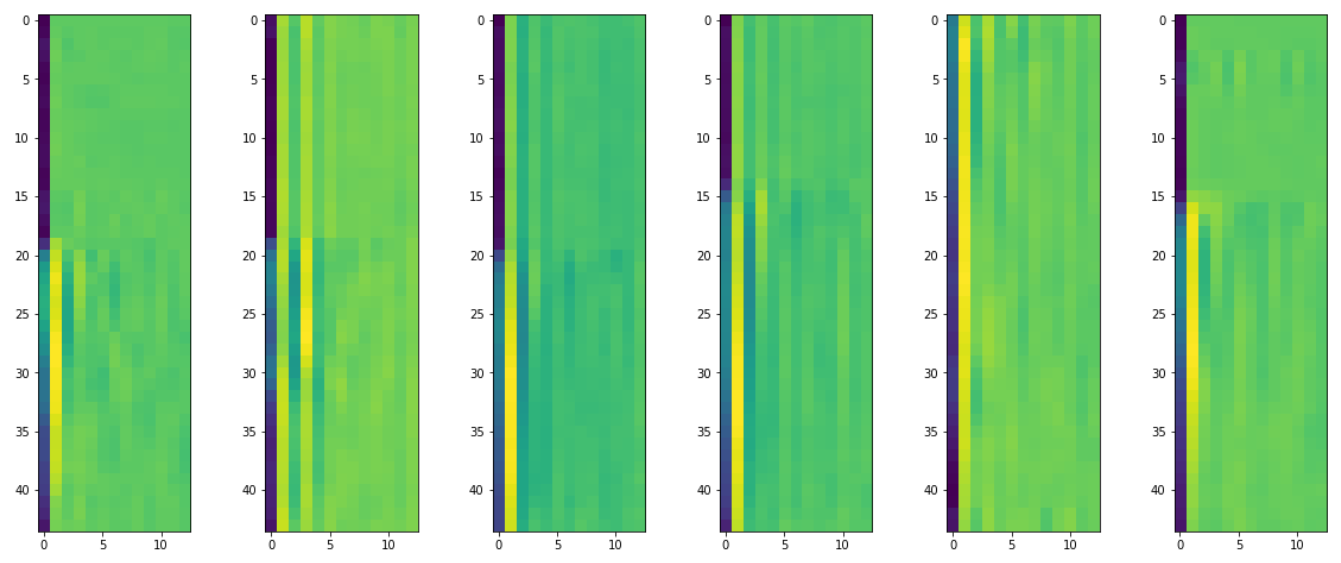
حلقه for اول فولدرهای داخل فولدر دیتاست را در نظر می‎گیرد و اسامی آن‎ها را وارد لیست mappings می‎کند در ادامه برای هر فولدر که شامل داده‎های یکی از کلاس‎های مورد نظر است یک حلقه for دیگر انجام می‎پذیرد، کار این حلقه استخراج فیچر mfcc یا spectrogram برای هر داده و ذخیره کردن آن در لیست‎های تعریف شده در دیکشنری data می‎باشد.

ضرایب mfcc به کمک تابع librosa.feature.mfcc() و ضرایب spectrogram به کمک تابع librosa.stft انجام می‎پذیرد. (Sample rate برابر با 22050 هرتز قرار داده شده است).

به دلیل حجم بالای محاسبات این دو فیچر به صورت غیر همزمان استخراج شده‎اند، ضرایب mfcc در فایل json و ضرایب spectrogram پس از محاسبه اندازه در فایل pickle ذخیره می‎شوند.

به علت محاسبات بالا به هنگام محاسبه ضرایب spectrogram از نیمی از داده‎ها به صورت رندوم صرف نظر شده و حدودا 4500 داده در این محاسبات شرکت می‎کنند.

دقت شود هنگام ذخیره سازی آرایه‎ها به لیست تبدیل شده باشند.

دیاگرام ضرایب MFCC را در شکل بالا مشاهده می­کنید.

**استفاده از ضرایب MFCC در شبکه MLP :**

پس از اکسترکت کردن ضرایب MFCC با استفاده از کد های زیر دیتا را می­خوانیم:

DATA\_PATH = "MFCC\_data.json"

SAVE\_MODEL\_PATH = "model\_mfcc.h5"

LEARNING\_RATE = 0.001

EPOCHS = 40

BATCH\_SIZE = 32

# This model has not yet been built. Build the model first by calling `build()` or by calling the model on a batch of data.

def load\_dataset(data\_path):

    with open(data\_path, "r") as fp:

        data = json.load(fp)

    X = np.array(data["MFCCs"])

    y = np.array(data["labels"])

    return X, y

سپس ورودی های X, y را با استفاده از تابع زیر به داده های train, validation, test تقسیم بندی می­کنیم:

def get\_data\_splits(data\_path, test\_size=0.1, test\_validation=0.1):

    # load the dataset

    X, y = load\_dataset(data\_path)

    # create train/ validation/ test sets

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=test\_size)

    X\_train, X\_validation, y\_train, y\_validation = train\_test\_split(X\_train,

y\_train, test\_size=test\_validation) # 0.1 of 0.9 -> 0.09

    # convert inputs from 2d to 3d arrays

    X\_train = X\_train[..., np.newaxis]

    X\_validation = X\_validation[..., np.newaxis]

    X\_test = X\_test[..., np.newaxis]

    return X\_train, X\_validation, X\_test, y\_train, y\_validation, y\_test

سپس با استفاده از تابع build\_model مدل شبکه MLP را می­سازیم. این تابع اندازه دیتا ورودی و همچنین learning rate را به عنوان ورودی می­گیرد.

def build\_model(input\_shape, learning\_rate):

    model = keras.Sequential()

    model.add(keras.layers.Flatten())

    model.add(keras.layers.Dense(128, activation="relu"))

    model.add(keras.layers.Dense(256, activation="relu"))

    model.add(keras.layers.Dropout(0.1))

    model.add(keras.layers.Dense(256, activation="relu"))

    model.add(keras.layers.Dense(64, activation="relu"))

    model.add(keras.layers.Dense(4, activation="softmax"))

    optimizer = keras.optimizers.Adam(learning\_rate=learning\_rate)

    model.compile(optimizer=optimizer,

loss="sparse\_categorical\_crossentropy",

metrics=["accuracy"])

    return model

همانطور که مشاهده می­شود این شبکه از نوع sequential با ۵ لایه Dense است که از تابع فعال­ساز relu استفاده کردیم و برای لایه آخر از ۴ نورون با تابع فعال­ساز softmax برای تشخیص چهار کلاس خروجی است. ترتیب خروجی ها بترتیب الفبای انگلیسی است ( Down, Left, Right, Up – بترتیب از چپ به راست ).

def main():

    X\_train, X\_validation, X\_test, y\_train, y\_validation,

y\_test = get\_data\_splits(DATA\_PATH)

    # build model

    input\_shape = (X\_train.shape[1], X\_train.shape[2], X\_train.shape[3]) # (segments, coefficients <13>, 1)

    model = build\_model(input\_shape, LEARNING\_RATE)

    # train the model

    history = model.fit(X\_train, y\_train, epochs=EPOCHS,

batch\_size=BATCH\_SIZE,

validation\_data=(X\_validation, y\_validation))

    # evaluate the model

    test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test)

    print(f"Test loss: {test\_loss}, test accuracy: {test\_accuracy}")

    model.save(SAVE\_MODEL\_PATH)

    #eval

    #

    print("evaling .......................")

    model.evaluate(X\_test,y\_test)

    preds = model.predict(X\_test)

    preds = np.array([np.argmax(x) for x in preds])

    str\_labels = ['down', 'left', 'right', 'up']

    pp\_matrix\_from\_data(preds, y\_test, cmap='Blues\_r', columns=str\_labels)

    return model, history

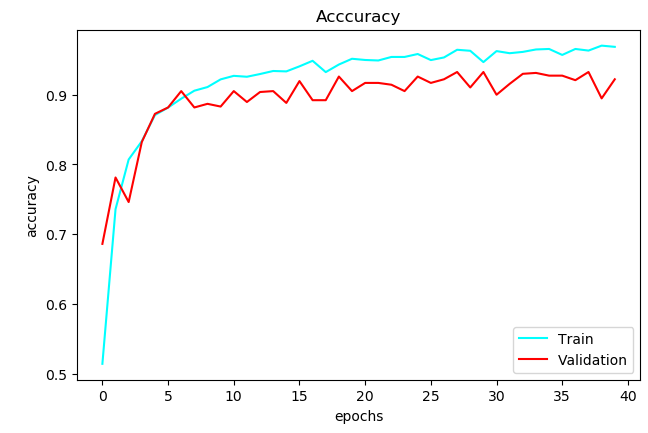
if \_\_name\_\_ == "\_\_main\_\_":

    model, history  = main()

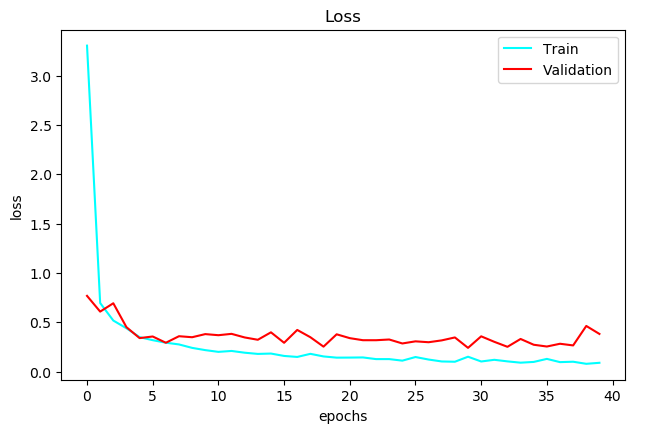
سپس با تابع main، دیتا را جدا می­کنیم، مدل را کامپایل و آن را برای دیتا های validation و test، بترتیب evaluate و predict می­کنیم که نتیجه مربوط به این مدل را در ادامه می­بینیم:

A picture containing text, font, receipt, music

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a graph

Description automatically generated with low confidence

**استفاده از spectrogram به عنوان تصویر و استفاده از شبکه‎های CNN:**

فیچرهای ذخیره شده در فایل spectrogram\_data.pkl را به کمک کد زیر خوانده و به numpy.array تبدیل می‎کنیم.

def load\_dataset(*data\_path*):

*with* open(DATA\_PATH, "rb") *as* f:

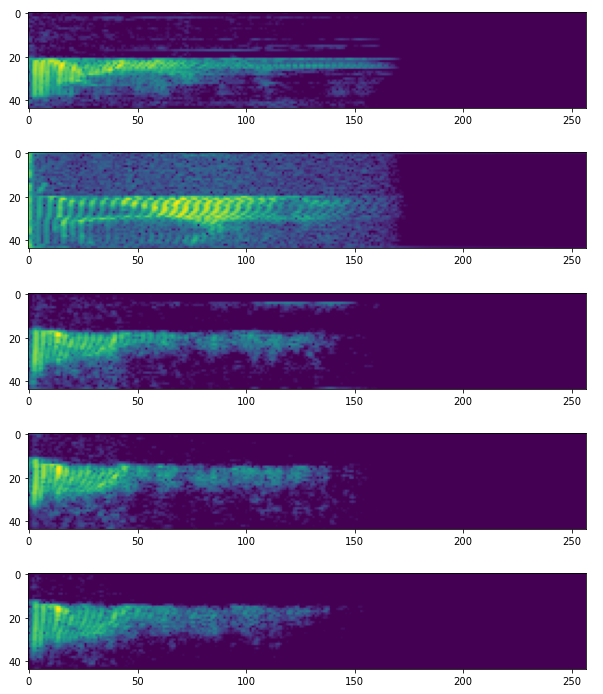
        data = pickle.load(f)

    X = np.array(data["spectrograms"])

    y = np.array(data["labels"])

*return* X, y

مشاهده چند نمونه از داده‎های spectrogram ذخیره شده؛



به کمک تابع train\_test\_split از کتابخانه sklearn داده‎ها را به سه قسمت train، validation، test تقسیم می‎کنیم. در ادامه داده‎ها سه بعدی شده تا در شبکه cnn وارد شوند.

def get\_data\_splits(*data\_path*, *test\_size*=0.1, *test\_validation*=0.1):

*# load the dataset*

    X, y = load\_dataset(data\_path)

*# create train/ validation/ test sets*

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,

*test\_size*=test\_size)

    X\_train, X\_validation, y\_train, y\_validation = train\_test\_split(X\_train,

y\_train,

*test\_size*=test\_validation)

*# 0.1 of 0.9 -> 0.09*

*# convert inputs from 2d to 3d arrays*

    X\_train = X\_train[..., np.newaxis]

    X\_validation = X\_validation[..., np.newaxis]

    X\_test = X\_test[..., np.newaxis]

*return* X\_train, X\_validation, X\_test, y\_train, y\_validation, y\_test

معماری شبکه استفاده شده شامل 6 لایه کانوولوشنی (به همراه maxpool) می‎باشد که به دنبال آن لایه Flatten و دو لایه Dense استفاده شده است. برای جلوگیری از اورفیت شدن بر داده‎های train از regularizer استفاده شده و در لایه Dense، drop out به کار رفته است.

از sparse\_categorical\_crossentropy به عنوان تابع loss و الگوریتم Adam برای optimizing استفاده کرده‎ایم.

def build\_model(*input\_shape*, *learning\_rate*):

    model = keras.Sequential()

*# layer 1*

    model.add(keras.layers.Conv2D(64, (3, 3), *activation*="relu",

*input\_shape*=input\_shape, *kernel\_regularizer*=keras.regularizers.l2(0.001)))

    model.add(keras.layers.BatchNormalization())

    model.add(keras.layers.MaxPool2D((3, 3), *strides*=(2, 2), *padding*="same"))

*# layer 2*

    model.add(keras.layers.Conv2D(32, (3, 3), *activation*="relu",

*kernel\_regularizer*=keras.regularizers.l2(0.001)))

    model.add(keras.layers.BatchNormalization())

    model.add(keras.layers.MaxPool2D((3, 3), *strides*=(2, 2), *padding*="same"))

*# layer 3*

    model.add(keras.layers.Conv2D(32, (3, 3), *activation*="relu",

*kernel\_regularizer*=keras.regularizers.l2(0.001)))

    model.add(keras.layers.BatchNormalization())

    model.add(keras.layers.MaxPool2D((3, 3), *strides*=(2, 2), *padding*="same"))

*# layer 4*

    model.add(keras.layers.Conv2D(32, (2, 2), *activation*="relu",

*kernel\_regularizer*=keras.regularizers.l2(0.001)))

    model.add(keras.layers.BatchNormalization())

    model.add(keras.layers.MaxPool2D((2, 2), *strides*=(2, 2), *padding*="same"))

    model.add(keras.layers.Flatten())

*# layer 5*

    model.add(keras.layers.Dense(8, *activation*="relu"))

    model.add(keras.layers.Dropout(0.3))

*# layer 6*

    model.add(keras.layers.Dense(4, *activation*="softmax"))

    optimizer = keras.optimizers.Adam(*learning\_rate*=learning\_rate)

    model.compile(*optimizer*=optimizer, *loss*="sparse\_categorical\_crossentropy",

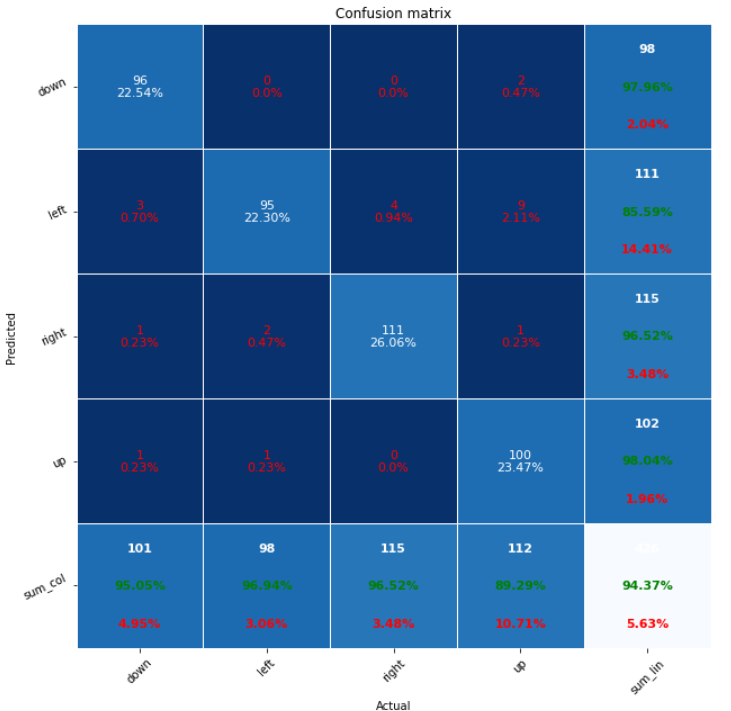
*metrics*=["accuracy"])

    model.summary()

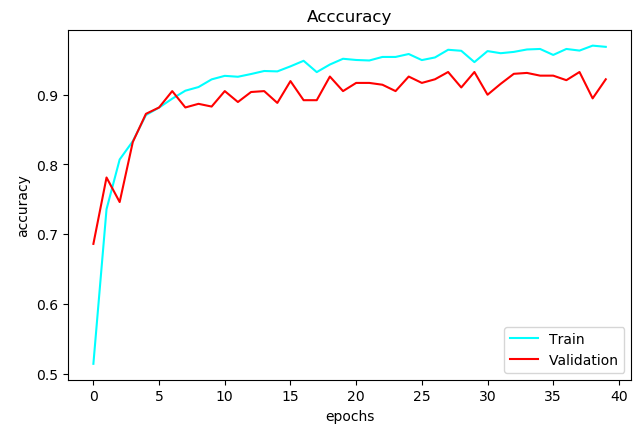
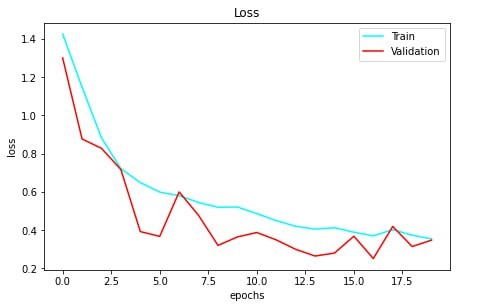
*return* model

مدل مجموعا 40 هزار پارامتر خواهد داشت نتیجه آموزش برای 40 ایپاک با نرخ یادگیری 001/0 برای داده‎های آموزش دارای دقت 8873/0، برای داده‎های صحت سنجی دارای دقت 9034/0 می‎باشد.

در ادامه به کمک تابع model.evaluate مشاهده می‎شود که دقت بر روی داده‎های تست، 9437/0 می‎باشد. Confusion matrix برای داده‎های تست رسم شده است.



نمودارهای loss و accuracy برای داده‎های آموزش و صحت سنجی؛



در مقایسه دو روش بررسی شده می‎توان به این نکته اشاره کرد که spectrogram به دلیل داده‎های زیادی برای هر سمپل تولید می‎کند شبکه پیچیده تری لازم دارد همچنین به دلیل ابعاد بالا محاسبات بالاتر می‎رود و زمان هر ایپاک حدود یک دقیقه می‎باشد که در مقایسه با فیچرهای mfcc و شبکه mlp که آموزش هر ایپاک چند ثانیه زمان لازم دارد بسیار بیشتر است.

**Dodge Lane :**

بازی DodgeLane با استفاده از کتابخانه‌ی PyGame طراحی و پیاده‌سازی شده است در طراحی این بازی اصول OOP مد نظر قرار گرفته اند. به طور کلی ساختار بازی از سه کلاس ایجاد شده است: Game، Player و Enemy. کلاس‌های Player و Enemy از کلاس pygame sprite وراثت گرفته‌اند. هواپیمای بازیکن توسط اسپرایت Player و هواپیمای بازیکن توسط اسپرایت Enemy ایجاد می‌شوند. برای تشخیص برخورد میان این دو نوع اسپرایت از pygame.sprite.collide\_mask استفاده شده است که با استفاده از alpha mask دو اسپرایت برخورد میان آن دو را بررسی می‌کند.

ظاهر شدن هواپیماهای دشمن توسط دو پارامتر و یک شرط کنترل می‌شود، پارامتر max\_num\_of\_enemies حداکثر تعداد هواپیماهای دشمن موجود در صحنه را مشخص می‌کند، در صورتی که تعداد هواپیماهای موجود از این مقدار کمتر باشد، با احتمال enemy\_spawn\_chance یک دشمن جدید در بالای صفحه ایجاد می‌شود، شرط ایجاد هواپیمای دشمن جدید نیز آن است که از تمام هواپیمای دشمن دیگر یک فاصله‌ی حداقلی داشته باشد تا دو هواپیمای دشمن در تماس با یکدیگر ظاهر نشوند.

# Spawn new enemies

               if len(self.enemies) < self.max\_num\_of\_enemies:

                   if random.random() < self.enemy\_spawn\_chance:

                       position = (random.randint(0, game.width - 80), 0)

                       # calculating minimum distance of new enemy to all

enemies to avoid spawning on another enemy

                       min\_dis = game.width

                       for x in self.enemies:

                           dis = np.linalg.norm(np.array(x.pos) –

np.array(position))

                           if dis < min\_dis:

                               min\_dis = dis

                       if min\_dis > 100:

                           new\_enemy = Enemy(position, 'enemy.png',

speed=self.enemies\_speed)

                           self.enemies.append(new\_enemy)

                           self.all\_sprites\_list.add(new\_enemy)

کد ۱: کنترل هواپیماهای دشمن موجود در صحنه

با ادامه پیدا کردن بازی، با توجه به زمان گذشته از شروع آن، پارامتر enemy\_spawn\_chance افزایش می‌یابد تا به مقدار ۱ (احتمال ۱۰۰ درصد) می‌رسد. سرعت دشمن‌ها نیز به مرور زمان افزایش می‌یابد و بیش‌تر و بیش‌تر می‌شود. موارد مذکور موجب افزایش سختی بازی به مرور زمان می‌شوند.

   def update\_difficulty(self):

       t = int(time.time() - self.init\_time)

       self.enemy\_spawn\_chance = min(self.initial\_enemy\_spawn\_chance + 0.1

\* (t / 1), 1)

       self.max\_num\_of\_enemies = self.initial\_max\_num\_of\_enemies + (t / 1)

       self.enemies\_speed = self.initial\_enemies\_speed + (t / 5)

کد ۲: تابع بروزرسانی پارامتر‌های مربوط به سختی بازی

بازیکن ۵ جان دارد، پس از ۵ بار برخورد، بازی به پایان می‌رسد و مدت زمانی که بازیکن توانسته است در بازی باقی بماند، به عنوان امتیاز وی در نظر گرفته می‌شود. در این هنگام با بهره‌گیری از کتابخانه‌ی easygui، نام کاربر پرسیده می‌شود و نام کاربر به همراه امتیاز وی (تعداد ثانیه‌هایی که کاربر توانسته در بازی باقی بماند) در یک پایگاه داده‌ی SQLite ذخیره می‌شود، همچنین تمام رکورد‌های موجود در پایگاه داده مرتب شده و براساس آن‌ها رتبه‌ی کاربر مشخص شده و به او نمایش داده می‌شود.

def insert\_new\_record(name, score):

   insert\_query = f"""INSERT INTO scores VALUES ("{name}", {score});"""

   sort\_query = """SELECT \* FROM scores ORDER BY score DESC;"""

   conn = create\_connection("scores.db")

   try:

       c = conn.cursor()

       c.execute(sort\_query)

       rows = c.fetchall()

       rank = len(rows)

       for i, (n, s) in enumerate(rows):

           if score > s:

               rank = i

               break

       c.execute(insert\_query)

       conn.commit()

   except Exception as e:

       print(e)

   return rank + 1

کد ۳: ارتباط با پایگاه داده، ذخیره‌ی امتیاز کاربر و تعیین رتبه‌ی او

**کنترل کردن بازی با کیبورد:**

برای کنترل کردن بازی علاوه بر فرامین صوتی می­توان از کیبورد نیز استفاده کرد بدین صورت که با استفاده از Arrow Key های کیبورد، هواپیما در جهت آن کلید حرکت می­کند همچنین با استفاده از کلید space هواپیما گلوله شلیک می­کند ( برای عدم بهم خوردن بالانس بازی ماکزیمم ۵ عدد گلوله می­توانند حاضر باشند تا موقعی که یکی نابود شود) که این گلوله ها در دو حالت نابود می­شوند: یکی زمانی که گلوله از تصویر خارج شود حالت دیگر زمانی که گلوله با هواپیما برخورد کند، در این حالت هم گلوله هم هوایپمای دشمن هر دو نابود می­شوند.

**کنترل کردن بازی توسط فرامین صوتی:**

برای کنترل کردن بازی توسط فرامین صوتی، از Redis استفاده شده است؛ همزمان با ماژول main که ماژول اصلی بازی است، ماژول predict نیز اجرا می‌شود، این ماژول مداوماً یک استریم صوتی از میکروفون دریافت کرده و توسط مدل train شده فرمان صوتی گفته شده را تشخیص می‌دهد، سپس در صورت تشخیص فرمان صوتی، آن را در یک key در redis ذخیره می‌کند، ماژول main نیز همواره در حلقه‌ی بروزرسانی خود، key مربوطه را از redis دریافت کرده و در صورت وجود مقدار (داده شدن فرمان صوتی) فرمان داده شده را در حرکت هواپیمای بازیکن اعمال می‌کند.

در ابتدای کد ماژول main یک ثابت با نام REDIS\_ENABLED وجود دارد که تعیین می‌کند اتصال ماژول با redis و نیز کنترل بازی توسط فرامین صوتی فعال باشد یا نه. لازم به ذکر است که برای استفاده از سرویس redis، این سرویس می‌بایست بر روی localhost بر روی پورت پیشفرض 6379 اجرا شده و قابل دسترسی باشد.

تصاویری از محیط بازی را در زیر مشاهده می­کنید:

A screenshot of a video game

Description automatically generated with medium confidence