دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده 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:

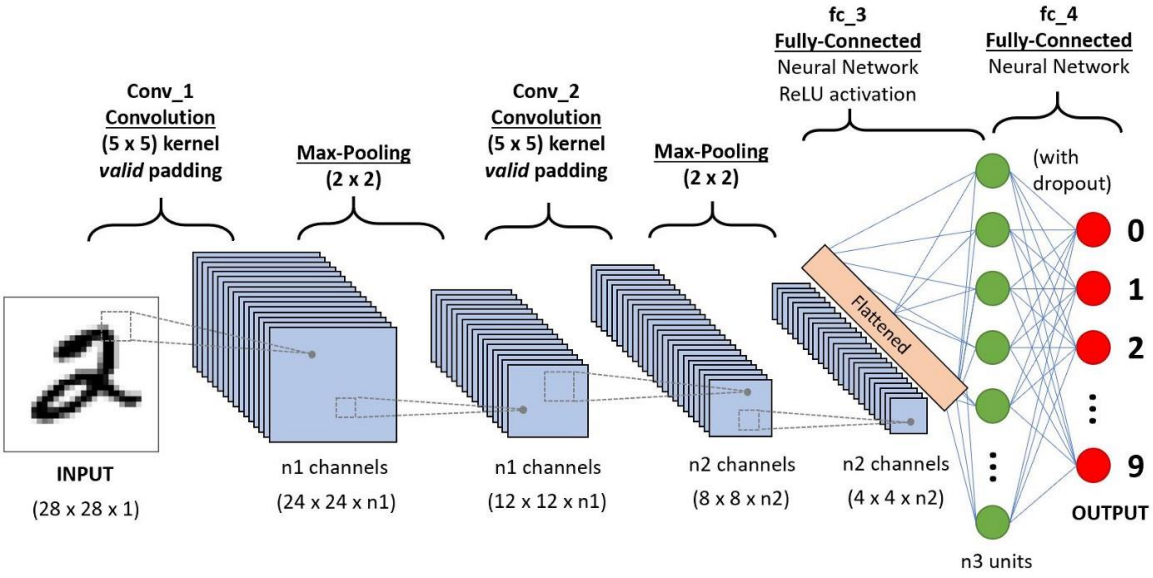
https://drive.google.com/file/d/1xoYZsE4EUadnmuRhex7ILZwG6v1SRtYR/view?usp=sharing

فهرست مطالب

پیش گزارش:

ساختار شبکه CNN :

شبکه کانولوشنی از لایه های اصلی زیر تشکیل شده است:

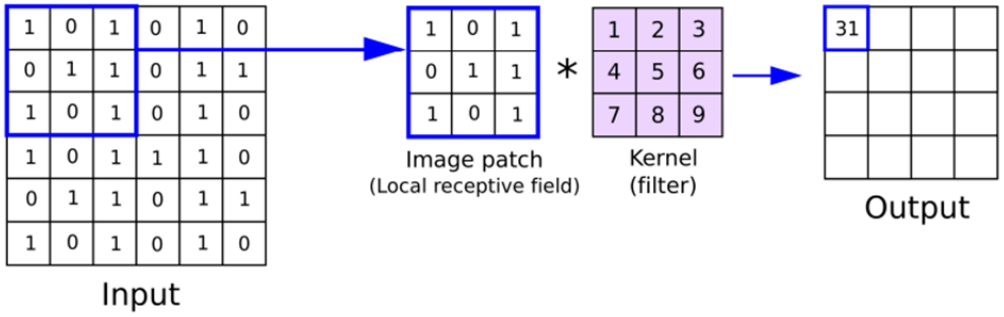


شکل 1) نمونه ای شبکه عصبی کانوولوشنی

- لایه کانولوشنی
- لایه ادغام
- لایه کاملا متصل

لایه کانولوشنی:

در این لایه عمل کانولوشن بر روی داده ورودی و با استفاده از تعدادی فیلتر انجام می شود. برای محاسبه هر درایه خروجی، ماتریس فیلتر بر روی ماتریس ورودی لغزانده می شود. عمل کانولوشن به این صورت تعریف می شود که ابتدا اولین عنصر فیلتر بر روی اولین عنصر ماتریس ورودی قرار می گیرد. سپس مجموع ضرب درایه های متناظر فیلتر با درایه های متناظز ماتریس ورودی محاسبه می شود. در نهایت فیلتر بر روی تصویر ورودی به اندازه پارامتر از پیش تعیین شده s به جلو برده می شود. با تکرار این مراحل ماتریس خروجی محاسبه می شود.



شکل 2) عملگر کانوولوشن

لایه ادغام:

یکی دیگر از لایه های شبکه کانولوشنی لایه ادغام است. این لایه هیچ پارامتر آموزشی ندارد. هدف این لایه کاهش ابعاد ماتریس ورودی ورودی و همزمان حفظ اطلاعات ارزشمند ورودی است. در این لایه ابتدا ماتریس با ابعاد از پیش تعیین شده k در نظر گرفته می شود. این پارامتر معمولا برابر با ۲ در نظر گرفته می شود. سپس با لغزاندن فیلتر بر روی ورودی اندازه ماتریس ورودی کاهش می یابد. یک نمونه از عملگرهایی که برای نمونه برداری در این لایه استفاده می شود، عملگر بیشینه است. در این حالت ماتریس از پیش تعیین شده بر روی داده ورودی لغزانده می شود و تنها بیشینه عناصری که در هر بخش قرار می گیرند را به عنوان خروجی در نظر می گیرد.

لایه کاملا متصل:

در این لایه یک شـبکه عصـبی کاملا متصـل قرار گرفته اسـت. در این لایه هدف مرتبط کردن ماتریس نهایی با خروجی های نهایی شبکه است. وزن های شبکه کاملا متصل از طریق پس انتشار خطا بدست می آید.

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, … می‌باشـد که برای این پروژه ما از لیبل های Up, Down, Left, Right اسـتفاده 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 ترتیب 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}")
```

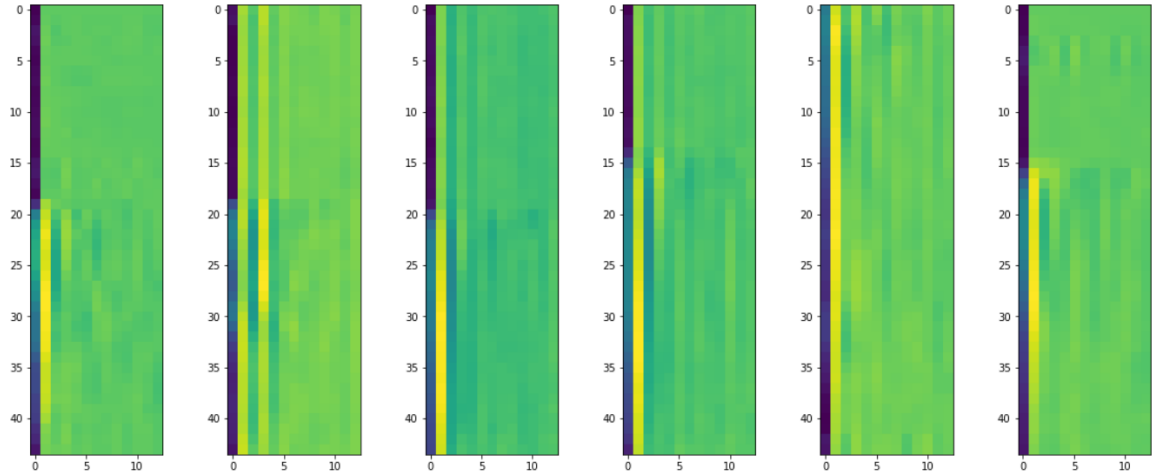
حلقه for اول فولدرهای داخل فولدر دیتاست را در نظر می‌گیرد و اسامی آن‌ها را وارد لیست mappings می‌کند در ادامه برای هر فولدر که شامل داده‌های یکی از کلاس‌های مورد نظر است یک حلقه for دیگر انجام می‌پذیرد، کار این حلقه استخراج فیچر mfcc یا spectrogram برای هر داده و ذخیره کردن آن در لیست‌های تعریف شده در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

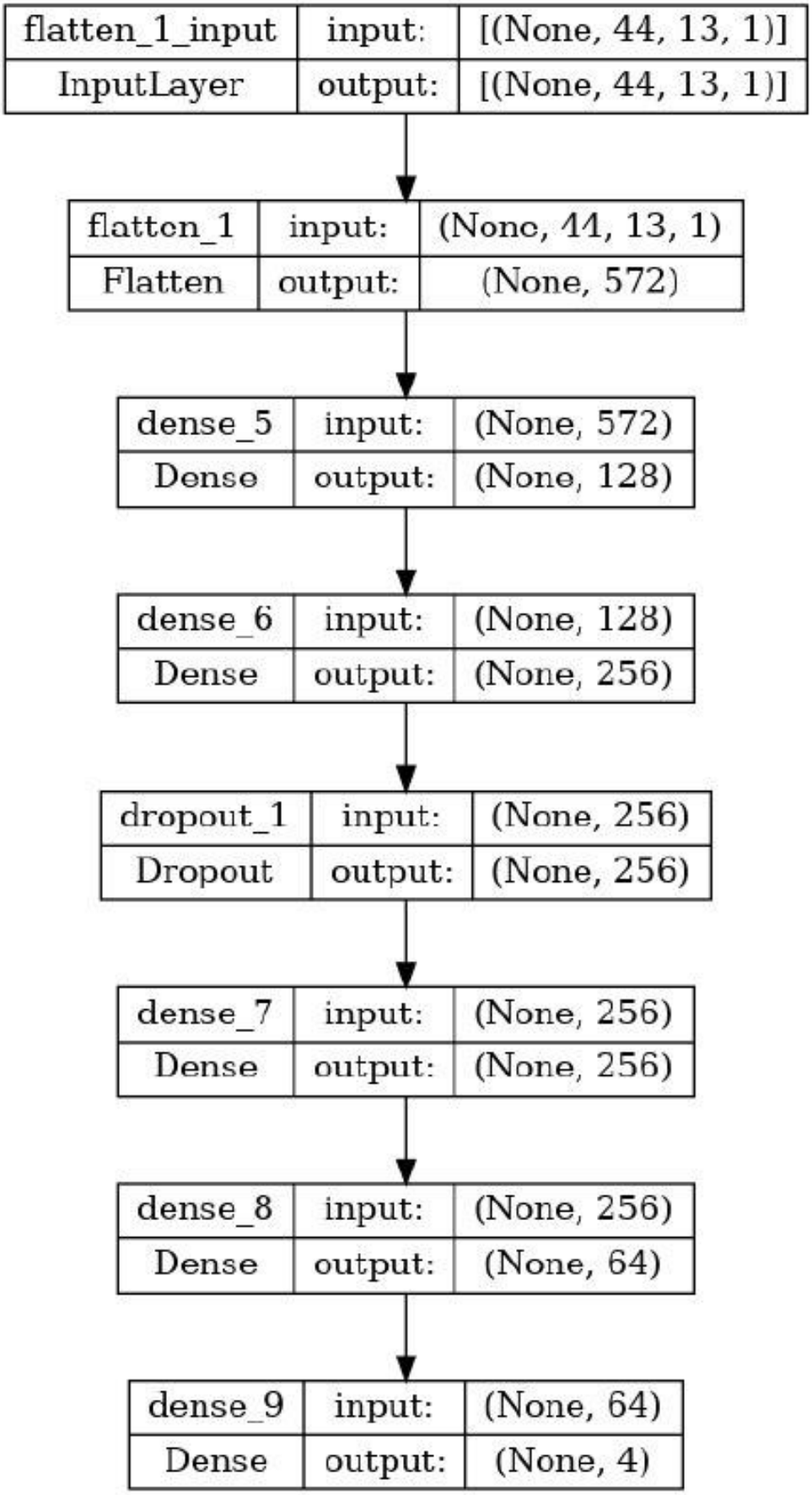
سپس با استفاده از تابع build_model مدل شبکه MLP را می‌سازیم. این تابع اندازه دیتا ورودی و همچنین learning rate را به عنوان ورودی می‌گیرد.

```
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
```

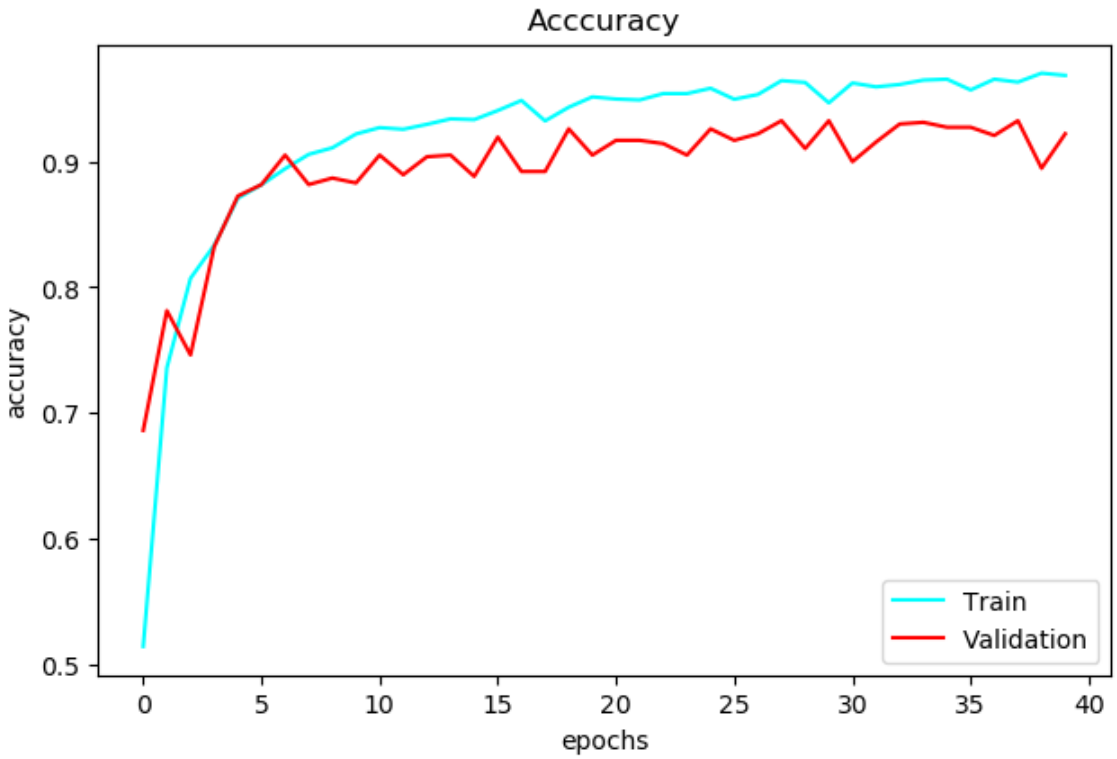
همانطور که مشاهده می‌شود این شبکه از نوع sequential با ۵ لایه Dense است که از تابع فعال‌ساز relu استفاده کردیم و برای لایه آخر از ۴ نورون با تابع فعال‌ساز softmax برای تشخیص چهار کلاس خروجی است. ترتیب خروجی ها بترتیب الفبای انگلیسی است (Down, Left, Right, Up – بترتیب از چپ به راست).

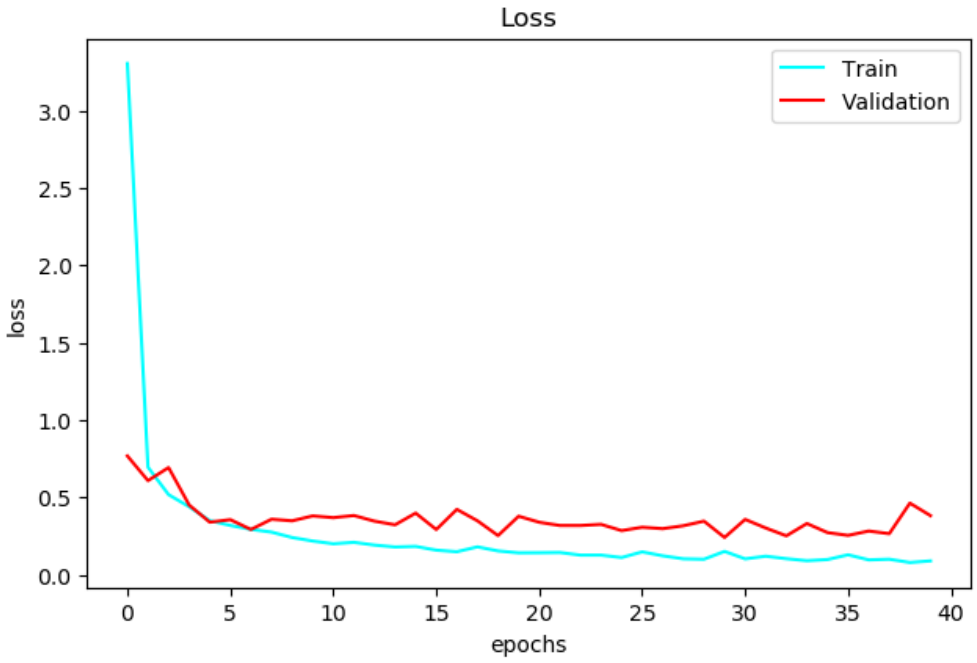
```
def main():

    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()
```

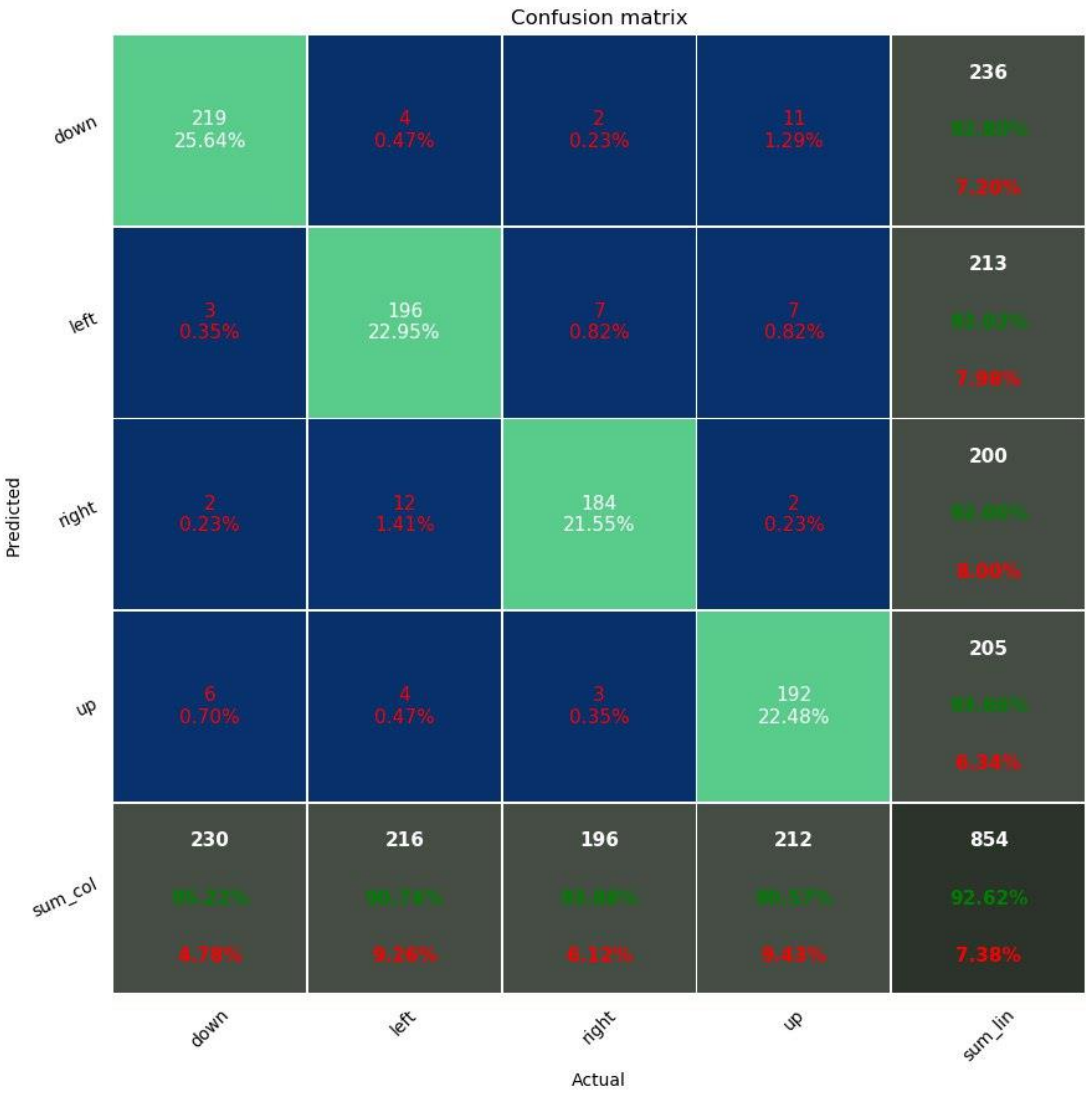
سپس با تابع main، دیتا را جدا می‌کنیم، مدل را کامپایل و آن را برای دیتا های validation و test، بترتیب evaluate و predict می‌کنیم که نتیجه مربوط به این مدل را در ادامه می‌بینیم:


flatten_1_input | input: | [(None, 44, 13, 1)]
InputLayer | output: | [(None, 44, 13, 1)]
flatten_1 | input: | (None, 44, 13, 1)
Flatten | output: | (None, 572)
dense_5 | input: | (None, 572)
Dense | output: | (None, 128)
dense_6 | input: | (None, 128)
Dense | output: | (None, 256)
dropout_1 | input: | (None, 256)
Dropout | output: | (None, 256)
dense_7 | input: | (None, 256)
Dense | output: | (None, 256)
dense_8 | input: | (None, 256)
Dense | output: | (None, 64)
dense_9 | input: | (None, 64)
Dense | output: | (None, 4)

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 flatten_1 (Flatten)          (None, 572)               0

 dense_5 (Dense)              (None, 128)               73344

 dense_6 (Dense)              (None, 256)               33024

 dropout_1 (Dropout)          (None, 256)               0

 dense_7 (Dense)              (None, 256)               65792

 dense_8 (Dense)              (None, 64)                16448

 dense_9 (Dense)              (None, 4)                 260

=================================================================
Total params: 188,868
Trainable params: 188,868
Non-trainable params: 0
_________________________________________________________________
```




Loss
Train
Validation
loss
epochs
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
5
10
15
20
25
30
35
40

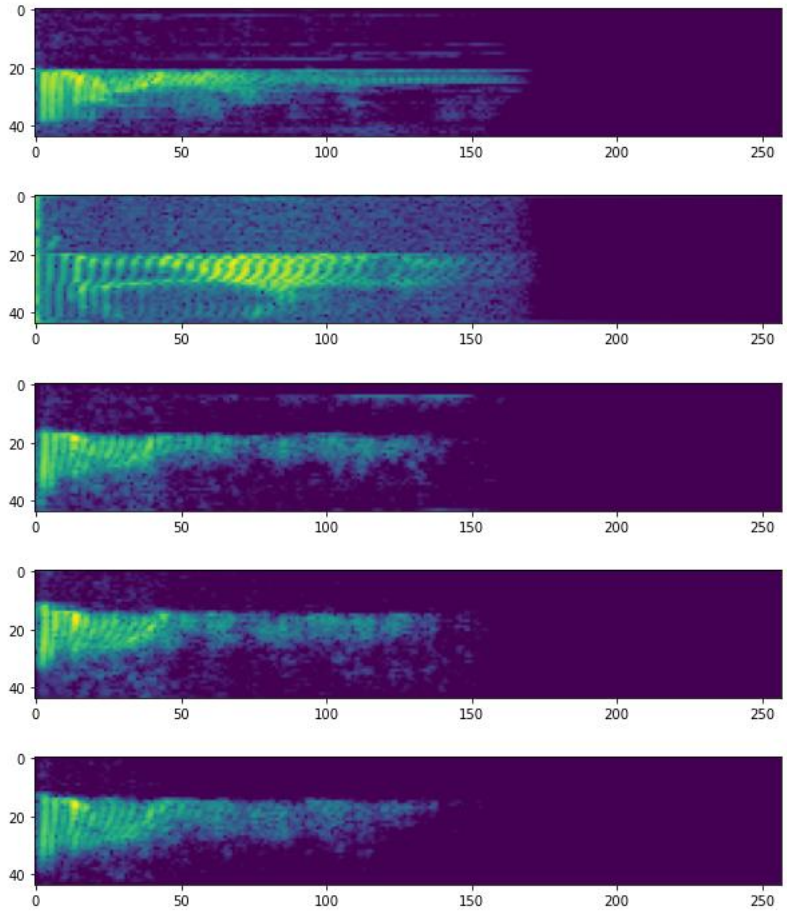

Confusion matrix
Predicted
Actual
down
left
right
up
sum_col
sum_lin
219
25.64%
4
0.47%
2
0.23%
11
1.29%
236
3
0.35%
196
22.95%
7
0.82%
7
0.82%
213
2
0.23%
12
1.41%
184
21.55%
2
0.23%
200
6
0.70%
4
0.47%
3
0.35%
192
22.48%
205
230
216
196
212
854
92.62%
7.38%

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.

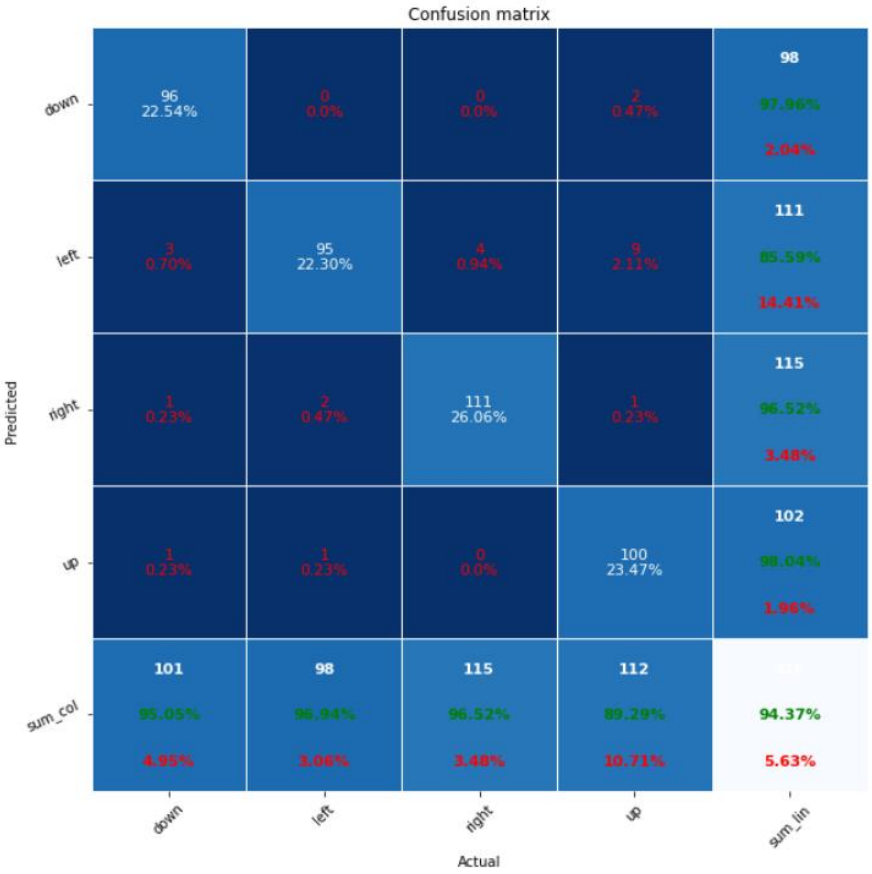
```
def build_model(input_shape, learning_rate):

    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"))
```

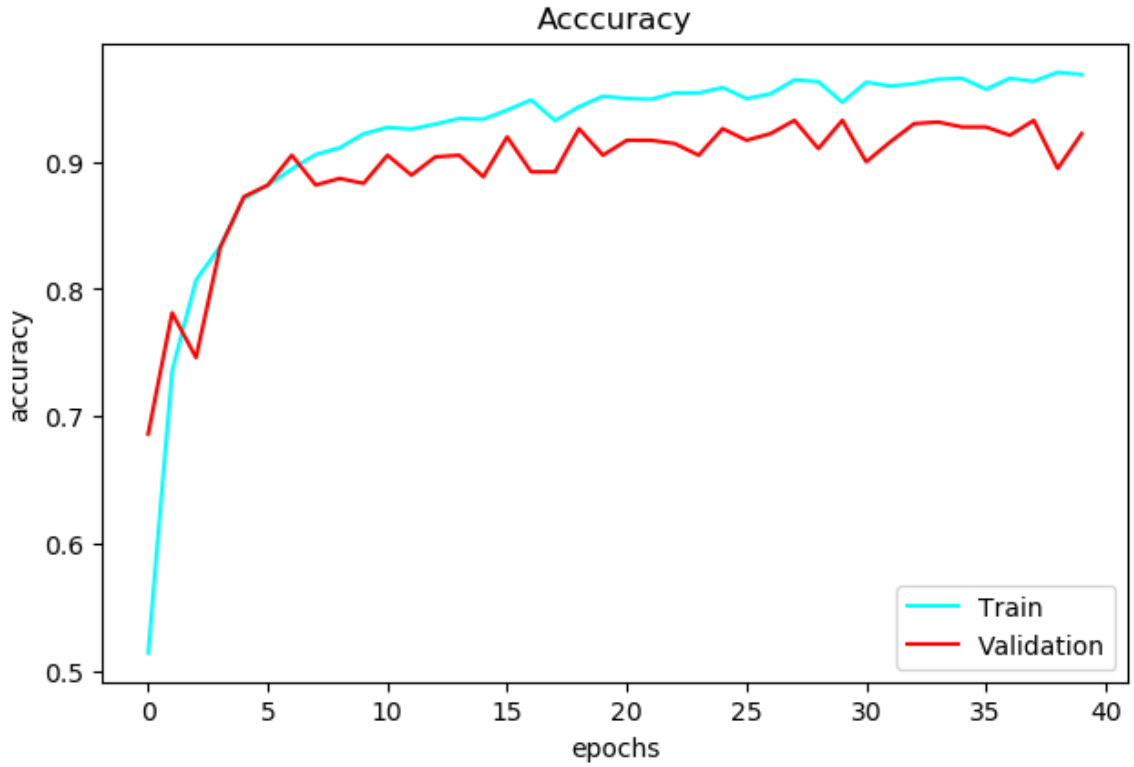
```
    optimizer = keras.optimizers.Adam(learning_rate=learning_rate)
                                        model.compile(optimizer=optimizer,
loss="sparse_categorical_crossentropy",
    metrics=["accuracy"])

    model.summary()

    return model
```

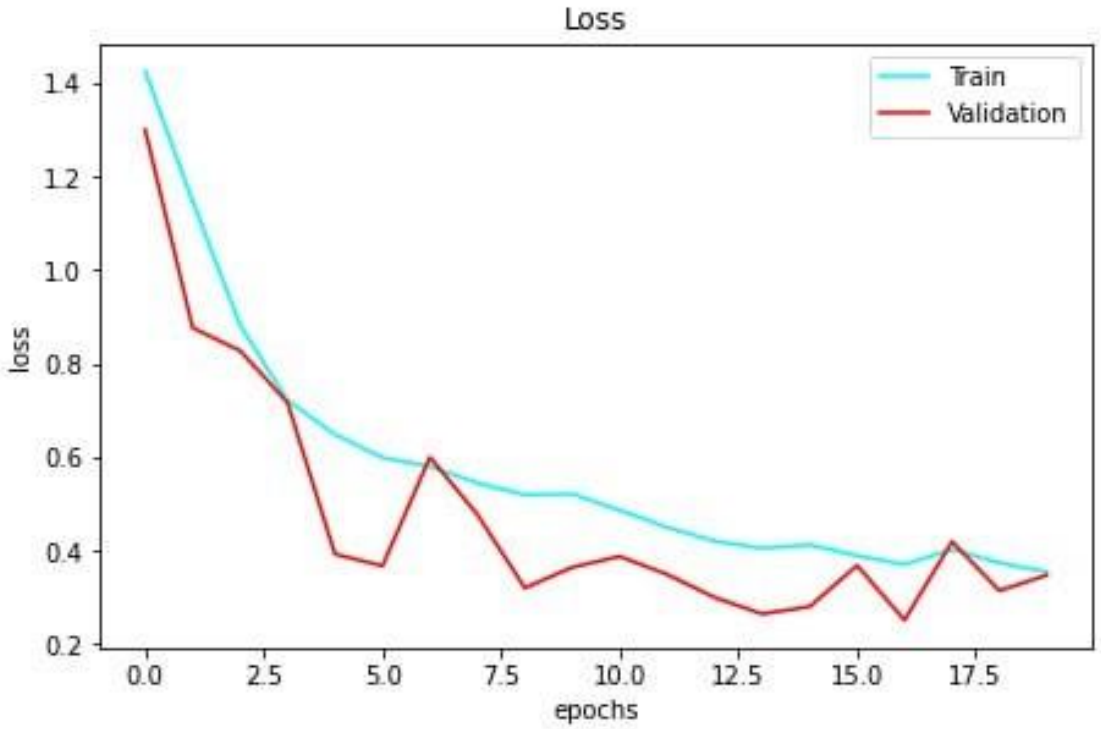
مدل مجموعا 40 هزار پارامتر خواهد داشت نتیجه آموزش برای 40 ایپاک با نرخ یادگیری 0/001 برای داده‌های آموزش دارای دقت 0/8873، برای داده‌های صحت سنجی دارای دقت 0/9034 می‌باشد.

در ادامه به کمک تابع model.evaluate مشاهده می‌شود که دقت بر روی داده‌های تست، 0/9437 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:

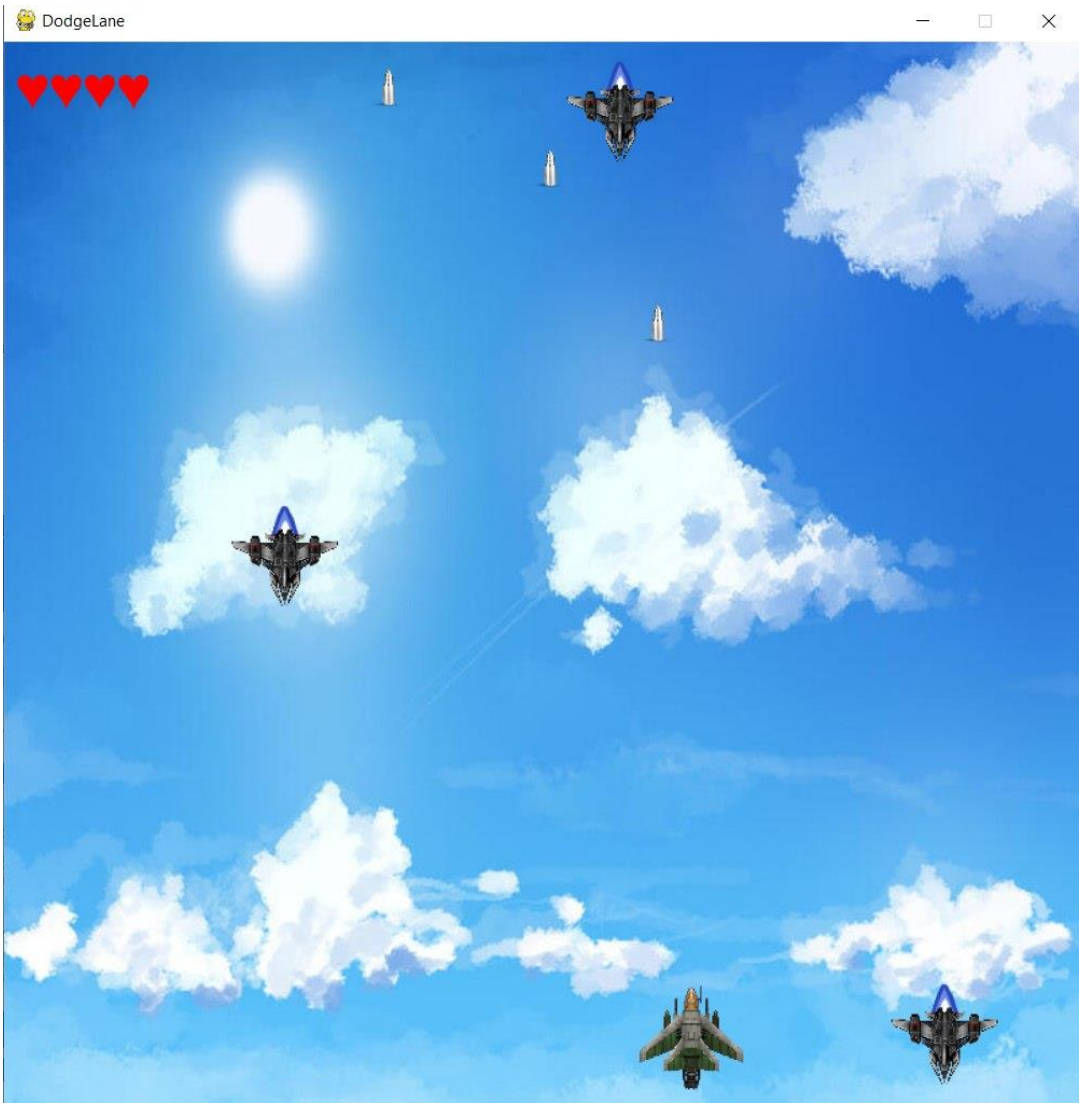
برای کنترل کردن بازی علاوه بر فرامین صوتی می‌توان از کیبورد نیز استفاده کرد بدین صورت که با استفاده از Arrow Key های کیبورد، هواپیما در جهت آن کلید حرکت می‌کند همچنین با استفاده از کلید space هواپیما گلوله شلیک می‌کند (برای عدم بهم خوردن بالانس بازی ماکزیمم ۵ عدد گلوله می‌توانند حاضر باشند تا موقعی که 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 هواپیمای دشمن هر دو نابود می‌شوند.

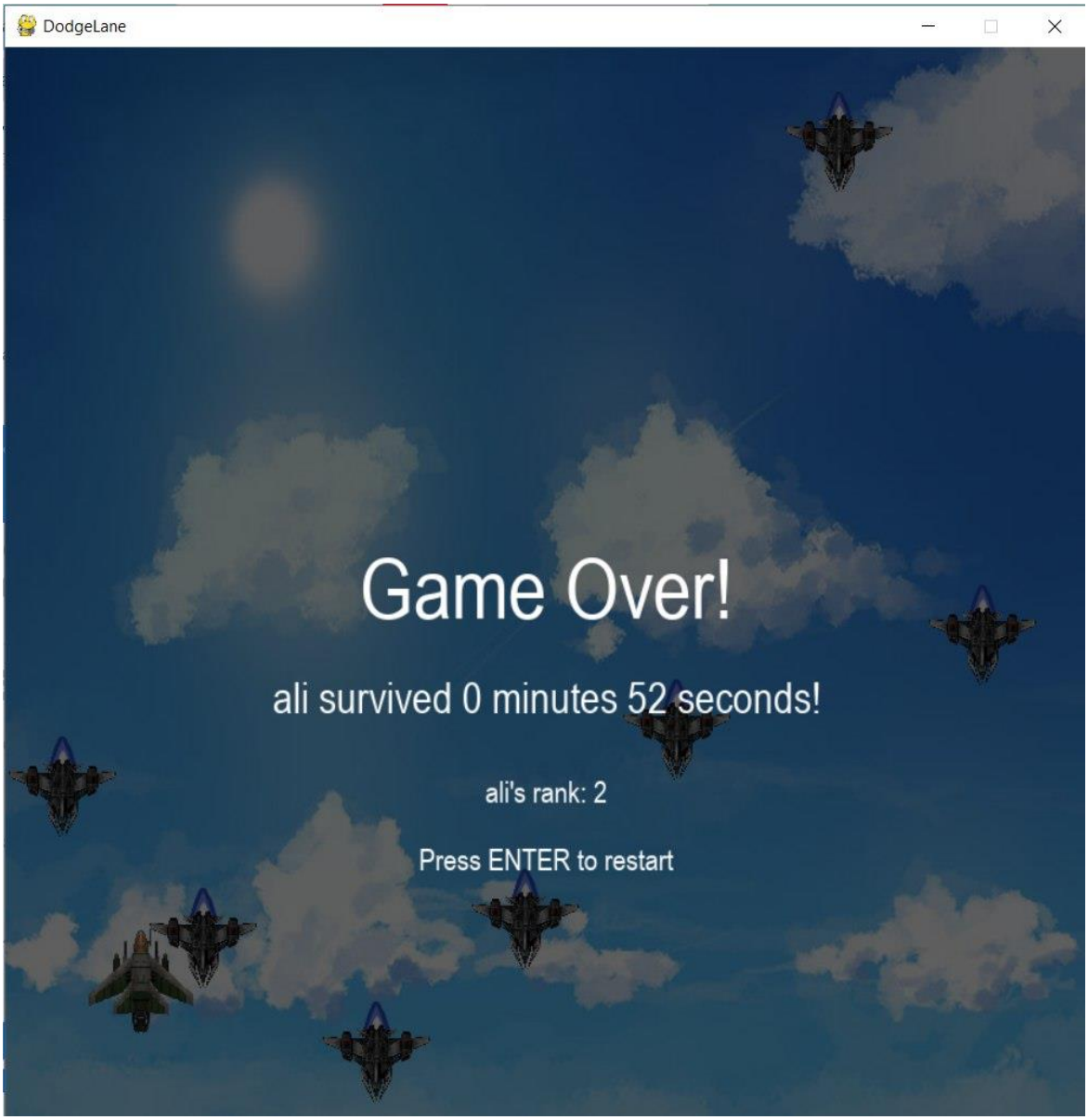
کنترل کردن بازی توسط فرامین صوتی:

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:




DodgeLane
Game Over!
ali survived 0 minutes 52 seconds!
ali's rank: 2
Press ENTER to restart