בס"ד

הפקולטה להנדסה

המעבדה לעיבוד אותות

Audio Neural Networks for Audio Pattern Recognition

סיווג קטעי אודיו באמצעות רשת נוירונים המתבססת על תמונת זמן-תדר ועל אות זמני

בן עטרה

מאיר וינברג

פרויקט שנה ד' לקראת תואר ראשון בהנדסה

מנחה: דניאל לוי

מנחה אקדמי: פרופ' שרון גנות

אוקטובר 2023

הפרויקט:

סיווג קטעי אודיו באמצעות רשת נוירונים המתבסס על תמונת זמן-תדר ועל אות זמני.

רקע לפרויקט:

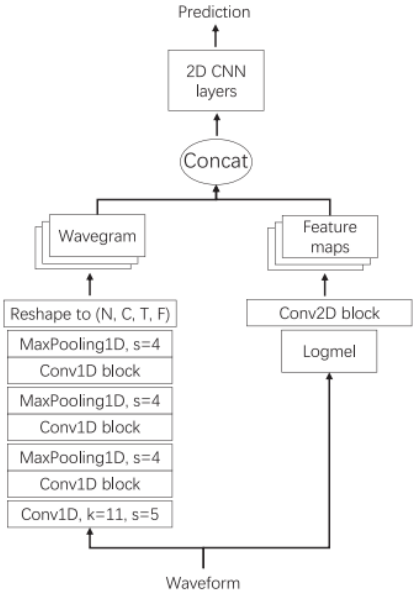
זיהוי דפוסי אודיו הוא נושא מחקר חשוב בתחום למידת מכונה, וכולל מספר משימות כגון תיוג אודיו, סיווג סצנות אקוסטיות, סיווג מוזיקה, סיווג רגשי דיבור וזיהוי אירועי קול.

לאחרונה, רשתות נוירונים יושמו כדי להתמודד עם בעיות זיהוי דפוסי אודיו. עם זאת, מערכות קודמות בנויות על מערכי נתונים ספציפיים עם משך זמן מוגבל. עם הזמן, בראייה ממוחשבת ועיבוד שפה טבעית, מערכות שהוכשרו מראש על מערכי נתונים בקנה מידה גדול התכללו היטב למספר משימות. בכל מקרה, קיים מחקר מוגבל על מערכות אימון מקדים בעלי מערכי נתונים בקנה מידה גדול לזיהוי דפוסי אודיו.

במאמר ממנו למדנו עבור הפרויקט, מציעים רשתות עצביות אודיו מאומנות מראש (PANNs) שהוכשרו על מערך הנתונים בקנה מידה גדול של AudioSet. רשתות אלה מועברות למשימות אחרות הקשורות לאודיו.

מטרת הפרויקט:

בפרויקט זה נלמד לעבוד עם רשתות נוירונים.
נראה איך שילוב של שני ייצוגים שונים של המידע לרשת יכולה לשפר את הביצועים הכוללים. כמו שניתן לראות בתמונה נבנה שתי רשתות שונות: הראשונה היא סיווג קטעי אודיו באמצעות רשת נוירונים המתבססת על אות זמני (רשת נוירונים אופיינית לסיווג זה), והשנייה היא רשת סיווג המתבססת על תמונת זמן-תדר. לבסוף ננסה לשלב בין הרשתות וננסה להגיע לביצועים מקסימליים.



נתבסס בפרויקט על הקורסים: למידת מכונה, למידה עמוקה ועיבוד ספרתי של אותות.

לו"ז לפרויקט:

תאריך	תיאור עבודה
סמסטר א'	איסוף מידע מתאים, בניית מערכת שתוכל לקלוט את המידע ככניסות לרשת
סמסטר ב'	בניית כל אחת מהרשתות, שילוב הרשתות, בדיקת הביצועים והסקת מסקנות

רשת ה – MobileNetV2 (תמונת זמן-תדר)

יצירת ה$Data$:

בקוד שלנו השתמשנו במידע מסוג $UrbanDataset$ שמכיל בתוכו 10 סוגים של קולות-

0 = air_conditioner
1 = car_horn
2 = children_playing
3 = dog_bark
4 = drilling
5 = engine_idling
6 = gun_shot
7 = jackhammer
8 = siren
9 = street_music

במידע הזה נשתמש לאורך כל הדרך בשתי הרשתות ובחיבור ביניהן.

קלטנו את הדאטה כאשר שרשנו את התיקייה למספר ה$fold$ המתאים לו ולשם הקובץ.

צורת המידע בתוך הקובץ:

	A	B	C	D	E	F	G	H
1	slice_file_name	fsID	start	end	salience	fold	classID	class
2	100032-3-0-0.wav	100032	0	0.317551	1	5	3	dog_bark
3	100263-2-0-117.wav	100263	58.5	62.5	1	5	2	children_playing
4	100263-2-0-121.wav	100263	60.5	64.5	1	5	2	children_playing
5	100263-2-0-126.wav	100263	63	67	1	5	2	children_playing
6	100263-2-0-137.wav	100263	68.5	72.5	1	5	2	children_playing
7	100263-2-0-143.wav	100263	71.5	75.5	1	5	2	children_playing
8	100263-2-0-161.wav	100263	80.5	84.5	1	5	2	children_playing
9	100263-2-0-3.wav	100263	1.5	5.5	1	5	2	children_playing
10	100263-2-0-36.wav	100263	18	22	1	5	2	children_playing
11	100648-1-0-0.wav	100648	4.823402	5.471927	2	10	1	car_horn
12	100648-1-1-0.wav	100648	8.998279	10.052132	2	10	1	car_horn
13	100648-1-2-0.wav	100648	16.699509	17.104837	2	10	1	car_horn
14	100648-1-3-0.wav	100648	17.631764	19.253075	2	10	1	car_horn
15	100648-1-4-0.wav	100648	25.332994	27.197502	2	10	1	car_horn
16	100652-3-0-0.wav	100652	0	4	1	2	3	dog_bark
17	100652-3-0-1.wav	100652	0.5	4.5	1	2	3	dog_bark
18	100652-3-0-2.wav	100652	1	5	1	2	3	dog_bark
19	100652-3-0-3.wav	100652	1.5	5.5	1	2	3	dog_bark
20	100795-3-0-0.wav	100795	0.19179	4.19179	1	10	3	dog_bark
21	100795-3-1-0.wav	100795	13.059155	17.059155	1	10	3	dog_bark
22	100795-3-1-1.wav	100795	13.559155	17.559155	1	10	3	dog_bark
23	100795-3-1-2.wav	100795	14.059155	18.059155	1	10	3	dog_bark
24	100852-0-0-0.wav	100852	0	4	1	5	0	air_conditioner
25	100852-0-0-1.wav	100852	0.5	4.5	1	5	0	air_conditioner
26	100852-0-0-10.wav	100852	5	9	1	5	0	air_conditioner
27	100852-0-0-11.wav	100852	5.5	9.5	1	5	0	air_conditioner
28	100852-0-0-12.wav	100852	6	10	1	5	0	air_conditioner
29	100852-0-0-13.wav	100852	6.5	10.5	1	5	0	air_conditioner
30	100852-0-0-14.wav	100852	7	11	1	5	0	air_conditioner
31	100852-0-0-15.wav	100852	7.5	11.5	1	5	0	air_conditioner
32	100852-0-0-16.wav	100852	8	12	1	5	0	air_conditioner
33	100852-0-0-17.wav	100852	8.5	12.5	1	5	0	air_conditioner
34	100852-0-0-18.wav	100852	9	13	1	5	0	air_conditioner
35	100852-0-0-19.wav	100852	9.5	13.5	1	5	0	air_conditioner
36	100852-0-0-2.wav	100852	1	5	1	5	0	air_conditioner
37	100852-0-0-20.wav	100852	10	14	1	5	0	air_conditioner
38	100852-0-0-21.wav	100852	10.5	14.5	1	5	0	air_conditioner
39	100852-0-0-22.wav	100852	11	15	1	5	0	air_conditioner
40	100852-0-0-23.wav	100852	11.5	15.5	1	5	0	air_conditioner
41	100852-0-0-24.wav	100852	12	16	1	5	0	air_conditioner
42	100852-0-0-25.wav	100852	12.5	16.5	1	5	0	air_conditioner
43	100852-0-0-26.wav	100852	13	17	1	5	0	air_conditioner
44	100852-0-0-27.wav	100852	13.5	17.5	1	5	0	air_conditioner
45	100852-0-0-28.wav	100852	14	18	1	5	0	air_conditioner

ניתן לראות שהאורך של כל דוגמה הינו שונה, ובחלק זה היינו צריכים לטפל בקוד שלנו.

חלק א'- ביצוע $STFT$

כמו שלמדנו, $STFT - short\ time\ Fourier\ transform$ הינה התמרה DFT כפי שהיא נראית מבעד לחלון כלשהו $v[n]$ הנתון לבחירתינו.

כאשר יש לנו אות שההרכב התדרי שלו משתנה בזמן (לדוגמה אות דיבור) נרצה לבצע התמרת פורייה במקטעים וכך נקבל רזולוציה יותר טובה של ההרכב התדרי בכל זמן.

התמרת הSTFT היא תלוית זמן ותדר (ספקטוגרמה).

הדרך לבצע זאת זה להכפיל את האות בחלון $v[n]$ (הפוך) מוזז סביב n באורך L_v :

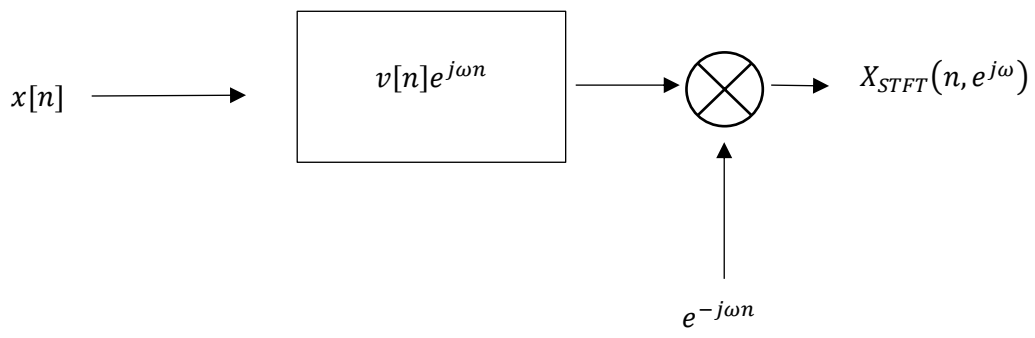
$$X_{STFT}(n, e^{j\omega}) = \sum_{m=-\infty}^{\infty} x[m]\, v[n-m] e^{-j\omega m} = \sum_{m=n-L_v}^{n} x[m]\, v[n-m] e^{-j\omega m}$$

ניתן לכתוב את זה בצורה הבאה:

$$X_{STFT}(n, e^{j\omega}) = e^{-j\omega n} \sum_{m=-\infty}^{\infty} x[m]\, v[n-m] e^{-j\omega m} e^{j\omega n}$$
$$= e^{-j\omega n} \sum_{m=-\infty}^{\infty} x[m]\, v[n-m] e^{j\omega(n-m)} =$$
$$e^{-j\omega n} \cdot \{x[n] * (v[n] e^{j\omega n})\}$$

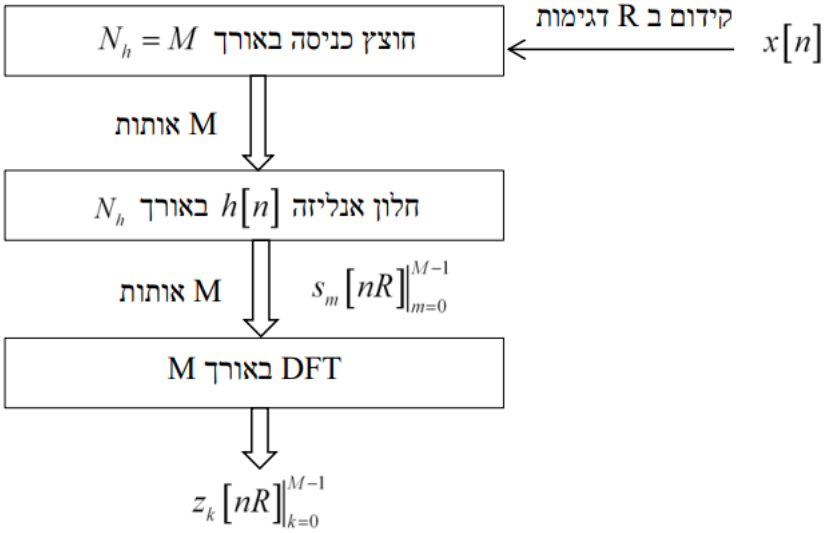
קיבלנו קונבולוציה עם חלון מוזז(בתדר). עבור כל ω_0 עבורו נרצה לחשב את ההתמרה נוכל לחשב אותה בקלות ע"י נוסחא זו.

באופן גרפי:



נזכור כי ככל שהחלון יותר ארוך רזולוציית התדר טובה יותר, וככל שהחלון יותר קצר רזולוציית הזמן טובה יותר.

להלן מערכת מפורטת לביצוע $STFT$:



לבסוף ברשת זו בחרנו את הנתונים שלנו להיות:

```
n_fft = 512
wlen = 512
overlap = int(wlen * 3 / 4)
n_hop = wlen-overlap
win = torch.hamming_window(wlen)
fs=16000
```

מה שאומר שמספר התדרים ואורך החלון זהים ולא נדרש קיפול מחזורי.

Mel sprectogram

הדרך בה אנו שומעים תדרים בצליל ידועה בשם 'גובה'. זו התרשמות סובייקטיבית של התדר. כלומר לצליל גבוה יש תדר גבוה יותר מאשר צליל נמוך. בני אדם אינם תופסים תדרים באופן ליניארי אלא אנו רגישים יותר להבדלים בין תדרים נמוכים יותר מאשר תדרים גבוהים.

לדוגמה, אם הקשבת לזוגות סאונד שונים באופן הבא:

$100Hz$ ו-$200Hz$

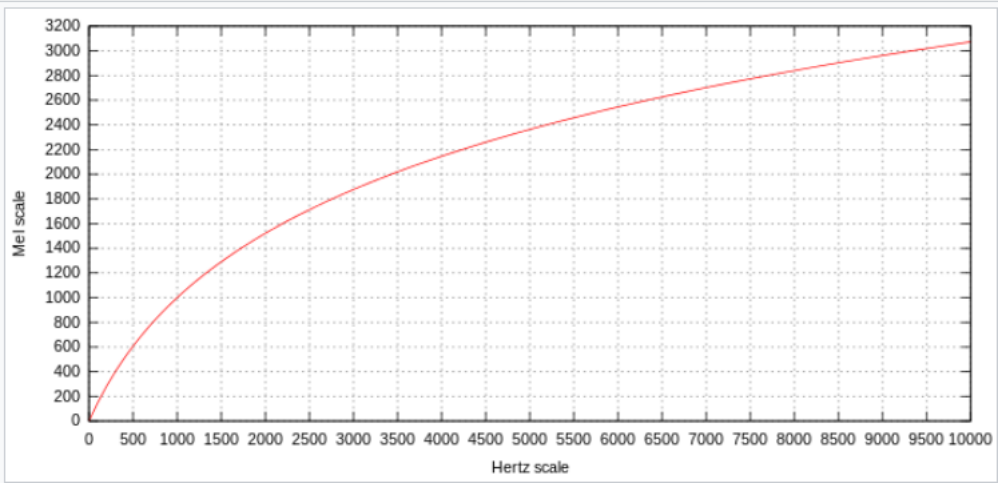
$1000Hz$ ו-$1100Hz$

למרות שבשני המקרים, הפרש התדרים בין כל זוג זהה לחלוטין ב-$100Hz$, הצמד ב-$100Hz$ ו-$200Hz$ יישמע רחוק יותר מהזוג ב-$1000Hz$ ו-$1100Hz$.

עם זאת, זה עשוי להיראות פחות מפתיע אם נבין שתדר $200Hz$ הוא למעשה כפול מ-$100Hz$ בעוד שתדר $1100Hz$ גדול ב10% מתדר $1000Hz$.

כלומר אנו שומעים תדרים בקנה מידה לוגריתמי ולא בקנה מידה ליניארי.

סולם מל פותח כדי לקחת זאת בחשבון על ידי עריכת ניסויים עם מספר רב של מאזינים. זהו סולם גובה הצליל, כך שכל יחידה נשפטת על ידי המאזינים להיות שווה בגובה הצליל מהשנייה.



$Mel\ sprectogram$ מבצעת שני שינויים חשובים ביחס לספקטוגרמה רגילה המתווה תדר מול זמן.

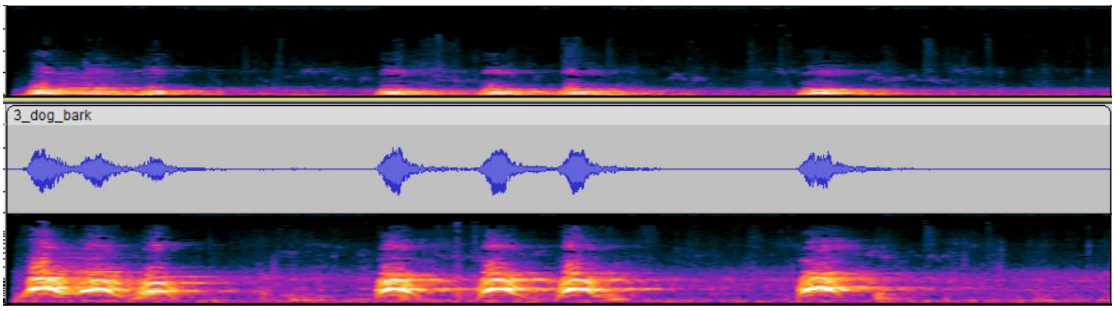
הוא משתמש בסולם מל במקום בתדר על ציר ה-y.

הוא משתמש בסולם דציבל במקום משרעת כדי לציין צבעים.

עבור מודלים של למידה עמוקה, אנחנו בדרך כלל משתמשים ב$Mel\ sprectogram$ ולא בספקטוגרמה פשוטה.

להלן דוגמה לשוני בין ספקטוגרמה רגילה, ל$Mel\ sprectogram$.

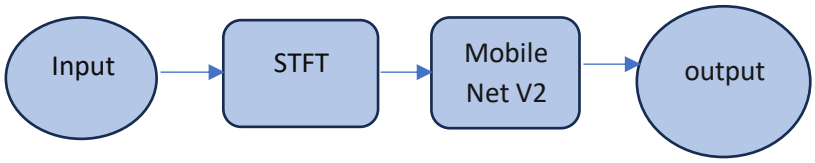
לדוגמה, נביחת כלב:



כאשר למעלה מוצגת ספקטוגרמה רגילה, באמצע קובץ שמע ולמטה Mel sprectogram.

תוצאה זאת הגיונית, כיוון שאנו רואים שבספקטוגרמה הרגילה התדרים הנמוכים הם הפעילים יותר, והMel נותן יותר מרווחים בין תדרים נמוכים (רואים זאת מהשיפוע המוצג בגרף למעלה).

בניית הרשת



פרמטרי הרשת

ניסינו לאפטם את התוצאות של הרשת שלנו, והרצנו כמה הרצות שונות של ההיפר פרמטרים בקוד שלנו-

1. $batch\ size$
2. $learning\ rate$
3. $optimization\ algorithm$

נציג בסוף את התוצאות לכל אחד מהערכים ונראה מה הגיע לתוצאות הטובות ביותר.

הכניסה לרשת

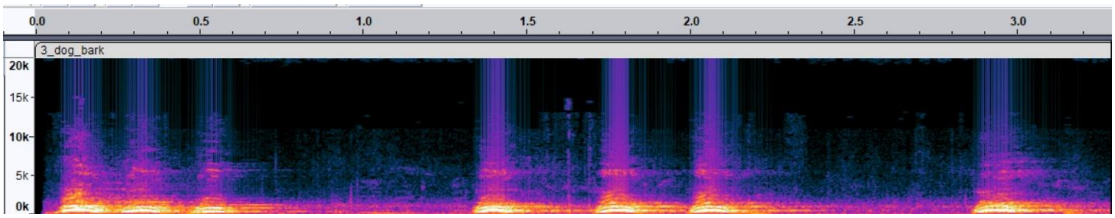
לפני הכניסה לרשת, ריפדנו את קבצי השמע בערכים קרובים ל0 (10^{-10}) לאורך אחיד, ורק לאחר מכן עשינו את ההתמרה באורך 512 תדרים. לפי מה שלמדנו בעיבוד ספרתי של אותות, ריפוד לפני ההתמרה רק משפר את הרזולוציה.

הכניסה לרשת הינה מטריצת ה-STFT שייצרנו. כפי שציינו המטריצה בגודל (1006×512). כידוע המימדים של הקלט משפיעים על מספר הפרמטרים שמאמנים ובכך משפיעים על הסיבוכיות של המודל.

סה"כ הגודל של המטריצה הסופית שהיא ה-input לרשת:

$$(batch\ size, channels, \text{frames}, \text{freaquencies})$$

דוגמא ל-input של הרשת:



שכבות הרשת

בחלק הראשון בחרנו לעבוד עם רשת מסוג $Moblie\ Net\ V2$.

$Moblie\ Net\ V2$ הינה רשת המיועדת למשימות ראיה ממוחשבת על תמונות. היא משתמשת במספר טכניקות ובלמידה עמוקה כדי לזהות מאפיינים מסוימים בתמונות.

הטכניקה המרכזית שמשמשת ב MobileNetV2
היא $depthwise\ seperable\ convolution$ -המשך חריגה. במקום לבצע קונבולוציה רגילה בין התמונה לבין מספר מסננים, היא מבצעת שני שלבים :הראשון הוא הקונבולוציה בין כל ערוץ (channel) בתמונה לבין מסנן קטן, ואחרי זה היא מבצעת קונבולוציה בין המסנן הקטן לבין התמונה שמגיעה מהשלב הראשון. זה מוריד את מספר הפרמטרים והחישובים משמעותית.

דבר נוסף, הרשת משתמשת במבני בלוקים מופרדים בשם $Inverted\ Residual\ Blocks$. כל בלוק מופרד כולל שלושת מסננים: קונבולוציה רגילה, של המשך חריגה, ושל קונבולוציה סופית. הם מאפשרים לשכפל את המאפיינים ולהגביר את היכולת המרחבית של המודל לזהות פרטים.

הרשת המוגדרת בתוך התוכנה מוציאה כפלט וקטור פיצ'רים בגודל $(batch\ size, 1000)$, ולאחר מכן נוסיף 3 שכבות $fully\ connected$ עם פונקציית אקטיבציה $ReLU$, כאשר לכל אחד נוסיף שכבת $dropout$ עם הסתברות p משתנה., כדי למנוע $overfitting$.

הקוד שמימשנו:

```python
class MobileNetV2_trained(nn.Module):


    def __init__(self,p=0.5):
        super(MobileNetV2_trained, self).__init__()
        self.model = models.mobilenet_v2(weights=True)
        for param in self.model.parameters():
            param.requires_grad = True  #update of params

        self.drop_layer1 = nn.Dropout(p=p)
        self.fc2 = nn.Linear(self.model.classifier[1].out_features, 512)
        self.drop_layer2 = nn.Dropout(p=p)
        self.fc3 = nn.Linear(512, 64)
        self.drop_layer3 = nn.Dropout(p=p)
        self.fc4 = nn.Linear(64, output_dim)


    def forward(self, x):

        x=torch.unsqueeze(x,1)
        x=x.repeat(1,3,1,1)
        x = self.model(x)
        x = self.drop_layer1(F.relu(x))
        x = self.drop_layer2(F.relu(self.fc2(x)))
        x = self.drop_layer3(F.relu(self.fc3(x)))
        x = self.fc4(x)
        return x

```

מוצא הרשת

כיוון שהרשת שלנו הינה רשת סיווג, מוצא הרשת הינו הלייבל המשוער של כל הקלטה.

הערה: כפי שציינו, במהלך הפרויקט בנינו שתי רשתות- רשת $Mobile\ Net\ V2, CNN$ ששתיהן מנסות לשערך בדרכים שונות את הלייבל הנכון של ההקלטה. נפרט תחילה על הרשת $Mobile\ Net\ V2,$ שמסווגת במישור הספקטרלי ונראה את התוצאות שקיבלנו, ואח"כ נפרט על הרשת שמבצעת CNN במישור הזמן. לבסוף ניקח את מוצאי שתי הרשתות ובאמצעות רשת מחברת ננסה לאפטם את התוצאות.

תוצאות הרשת

שלב אימון הרשת

כדי לאמן את הרשת עבדנו עם פונקציית מחיר $BCEWithLogitsLoss$.

הקוד שמימשנו:

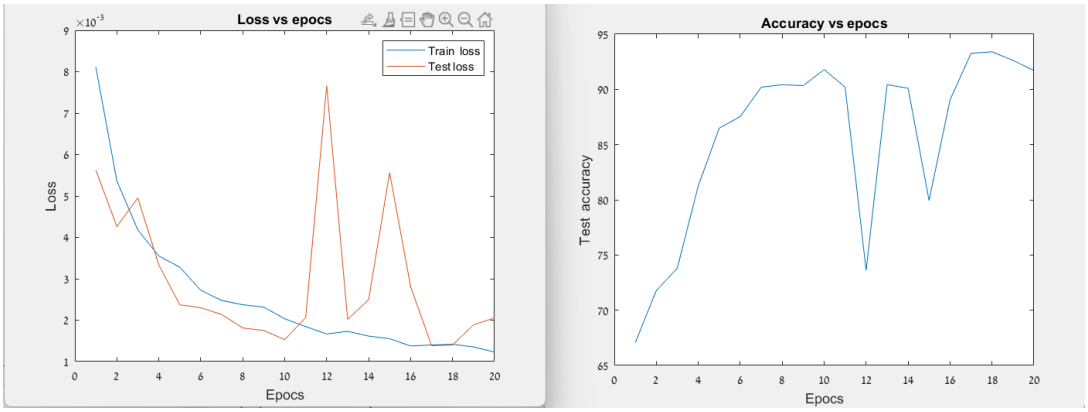
```
criterion = nn.BCEWithLogitsLoss()
network = MobileNetV2_trained()
network.to(device)
#optimizer = optim.SGD(network.parameters(), lr=learning_rate,momentum=momentum)
optimizer = optim.Adam(network.parameters(), lr=learning_rate)
train_losses = []
train_counter = []
test_losses = []
test_counter = [i*len(train_loader.dataset) for i in range(n_epochs + 1)]
Run Cell | Run Above | Debug Cell
#%%
def train(n_epoch):
  loss_all=0
  network.train()
  for batch_idx, (data, target) in enumerate(train_loader):
    data=data.to(device)
    target=target.to(device)
    optimizer.zero_grad()
    output = network(data)
    loss = criterion(output, target.float())
    loss.backward()
    torch.nn.utils.clip_grad_norm_(network.parameters(),1)
    optimizer.step()
    if batch_idx % log_interval == 0:
      print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(n_epoch, batch_idx * len(data), len(train_
                                                                      batch_idx / len(train_loader), loss.item()
      train_losses.append(loss.item())
      train_counter.append((batch_idx*64) + ((n_epoch-1)*len(train_loader.dataset)))

    loss_all+=loss.item() /len(train_loader.dataset)
  return loss_all
```

איפטמנו כל משתנה בנפרד- קודם את ה$learning\ rate$ לאחר מכן את ה$droupout$, $batch\ size$ ולבסוף את $optimization\ algorithm$.

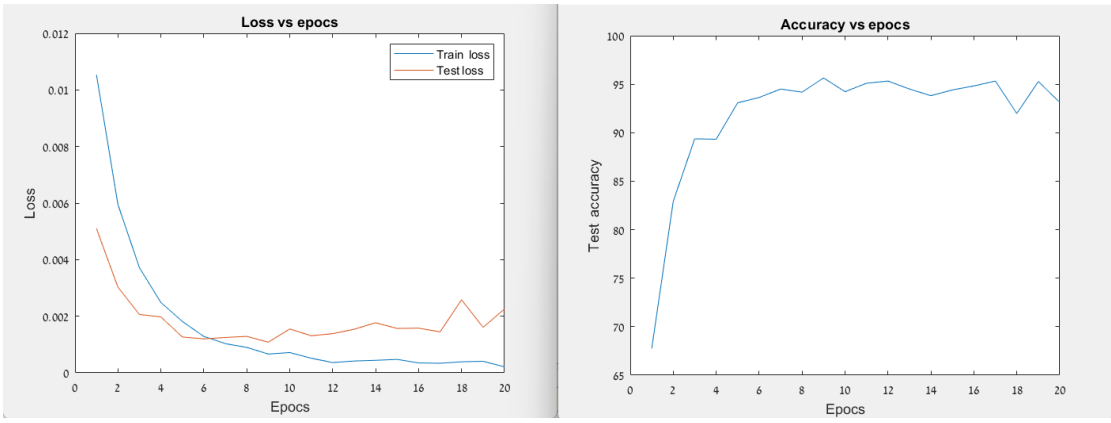
להלן התוצאות שהתקבלו:

$learning\ rate - 0.001$

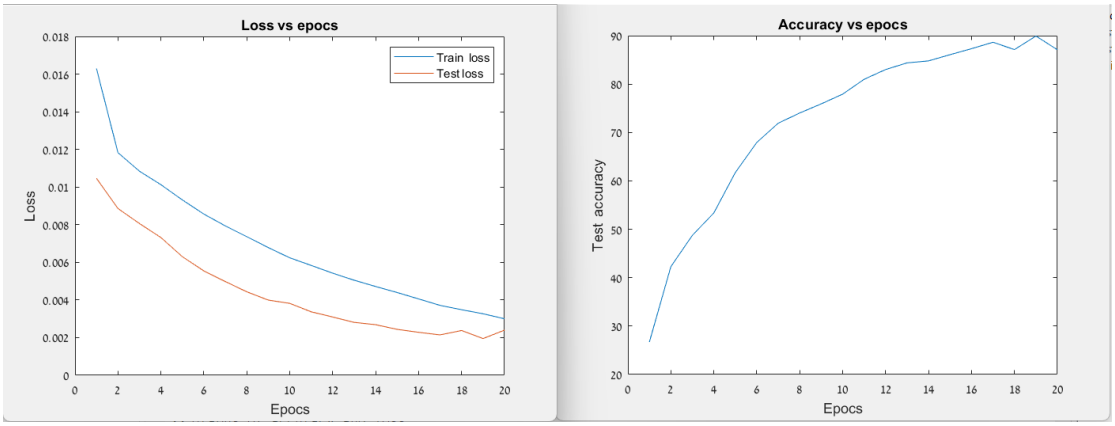


נראה כי הצעדים גדולים מדי, ועל כן התוצאות לא יציבות בכלל.

$learning\ rate - 0.0001$

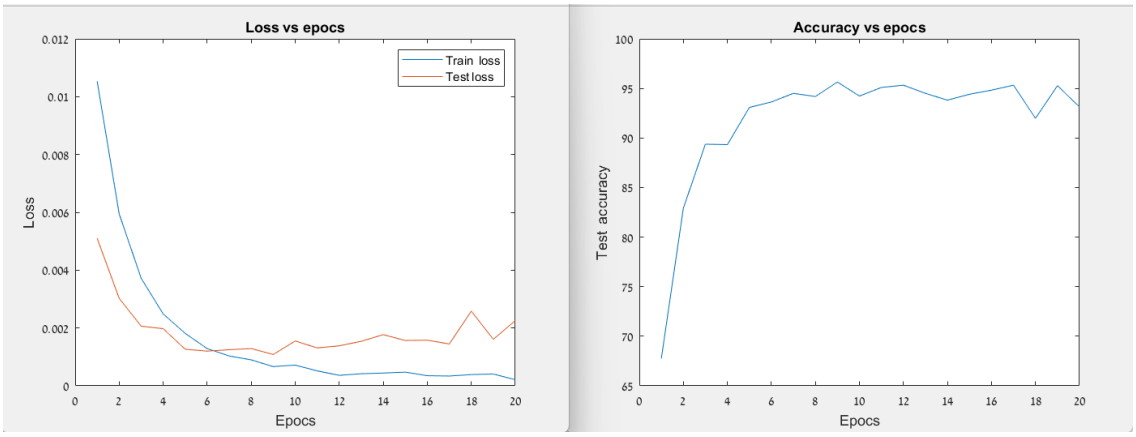


$learning\ rate - 0.00001$

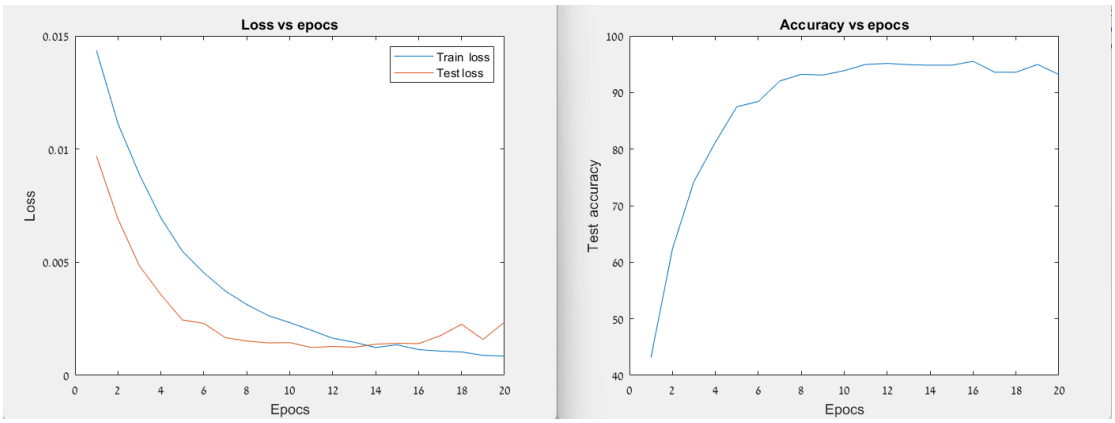


התוצאות הכי טובות היו ב $learning\ rate = 0.001$ וכעת נוסיף $dropout$

$dropout = 0.5$



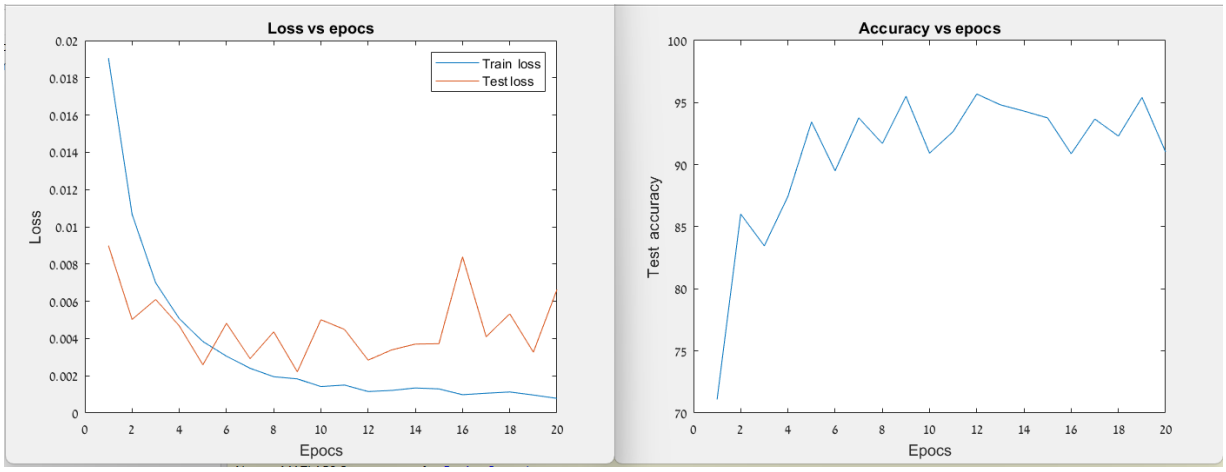
$dropout = 0.7$



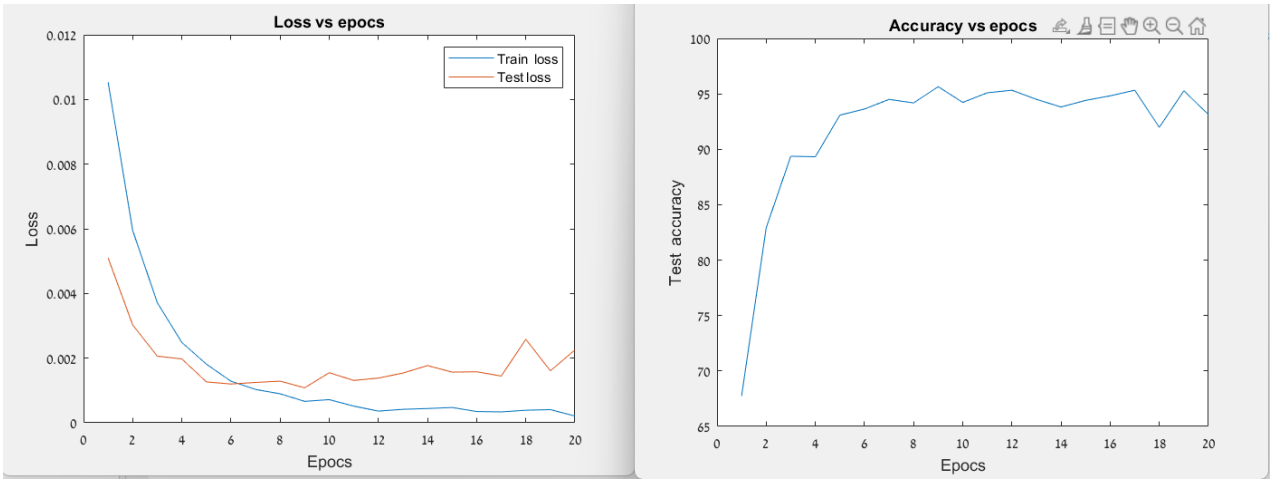
התוצאות טובות ויציבות יותר ב $dropout = 0.7$ ולכן נבחר ערך זה.

כעת נשנה את ה$batch\ size$.

$batch\ size = 16$



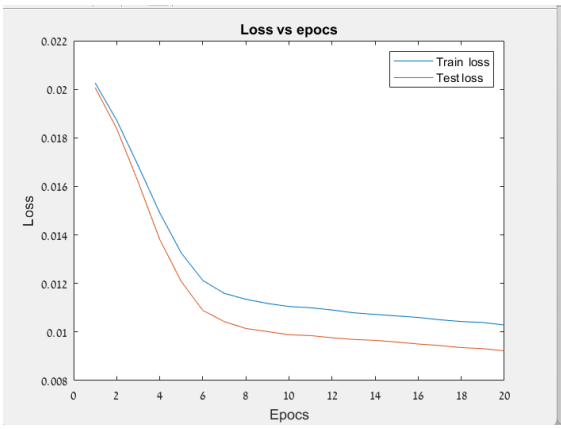
$batch\ size = 32$

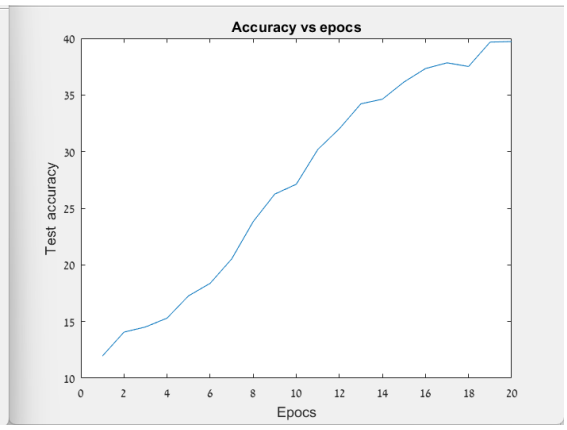


התוצאות טובות ויציבות יותר ב $batch\ size = 32$ ולכן נבחר ערך זה

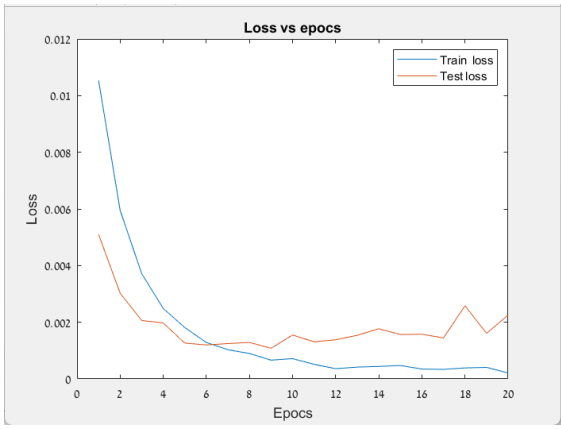
כעת נשנה את פונקציית ה$loss$

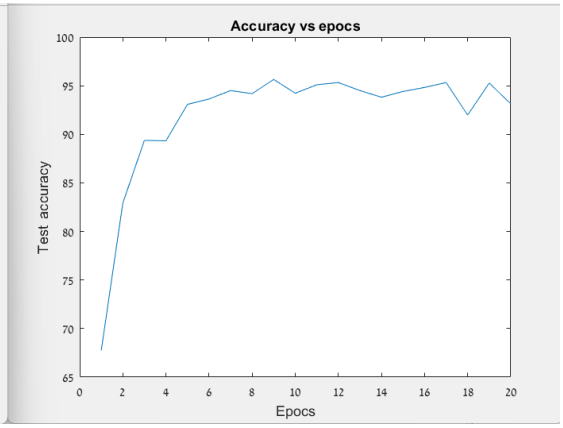
SGD





$ADAM$





כמובן נבחר ב$ADAM$ כיוון שמניב תוצאות טובות משמעותית.

ניתן לראות שככל שמספר ה-epoch-ים עולה כך הדיוק של ה$test$ וה-loss של סט האימון משתפרים.

הגענו לבסוף לדיוק של 93%.

רשת ה– One-dimensional CNN

הסבר כללי על הרשת

ראשית נסביר על רשת הקונבולוציה החד-מימדית ולמה היא משמשת באופן כללי:

רשת קונבולוציה חד-ממדית (One-dimensional CNN) היא סוג של ארכיטקטורת רשת נוירונים המיועדת בעיקר לעיבוד וניתוח נתונים רציפים חד-ממדיים, כגון סדרות מידע שתלויות בזמן, אותות אודיו, נתוני טקסט (כאשר מיוצגים כרצף של מילים או תווים), או כל נתונים עם מבנה ליניארי.

בעוד שרשתות -CNN מוכרות משמשות בדרך כלל למידע בצורת תמונה עם רשתות דו-ממדיות, CNN חד-מימדיים מותאמים לעבוד ביעילות עם רצפים.

מכיוון שבחלק זה ננסה לסווג את קטעי האודיו, נשתמש ברשת קונבולוצייה חד מימדית.

כעת נסביר על מבנה סטנדרטי של CNN חד מימדי, ושיקולים חשובים בעת בניית רשתות כאלו:

שכבות קונבולציה: בדיוק כמו ב-CNN דו-ממדיים, CNN חד-ממדיים משתמשים בשכבות קונבולוציוניות כדי לחלץ תכונות מנתוני הקלט. במקרה החד-מימדי, מסננים קונבולוציוניים אלה מועברים לאורך הרצף של הקלט כדי ללכוד דפוסים מקומיים. למסננים אלה יש רוחב קבוע, אך הם יכולים ללמוד דפוסים שונים כשהם נעים על פני הקלט.

Pooling layers: לאחר קונבולוציה, שכבות הPooling משמשות לעתים קרובות כדי לדגום את התכונות שחולצו. Max-pooling or average-pooling מוחל על חלונות קטנים כדי להפחית את הממדיות של מפות התכונות. Pooling עוזר לשמור מידע חשוב תוך הפחתת העומס החישובי (פחות מידע לזכור).

פונקציות אקטיבציה: פונקציות אקטיבציה, כגון ReLU (Rectified Linear Unit), מיושמות לאחר קונבולוציה וPooling כדי להכניס את חוסר הלינאריות למודל. זה מאפשר לרשת ללמוד דפוסים ויחסים מורכבים בתוך הנתונים.

מסננים קונבולוציוניים מרובים: עבור רשתות קונבולוציה חד-מימדיות משתמשים בדרך כלל במספר מסננים בכל שכבה קונבולוציונית. כל מסנן לומד דפוסים שונים מנתוני הקלט. עומק המסנן קובע את מספר המסננים בשכבה, והפרמטר הזה הוא היפרפרמטר שניתן למצוא את המיטבי עבורו הרשת עובדת הכי טוב.

Stride and Padding: Stide קובע את גודל הצעד של המסנן הקונבולוציוני כשהוא נע על פני הקלט (כמה מדלגים על מידע). בנוסף ניתן להשתמש בריפוד כדי לשלוט בגודל של מפות תכונת הפלט. ניתן לכוונן את הפרמטרים הללו כדי להשפיע על גודל הפלט ורמת חילוץ התכונות.

שיטוח: לאחר קונבולוציה וPooling , משטחים את מפות התכונות לרוב לוקטור במימד אחד. וקטור זה מועבר לאחר מכן דרך שכבה אחת או יותר מחוברת במלואה (שכבות צפופות) כדי לבצע תחזיות או לבצע ניתוח נוסף.

שכבת הפלט: שכבת הפלט תלויה במשימה הספציפית. עבור משימות סיווג, יש בדרך כלל כמות נוירונים כמו כמות המחלקות, ופונקציית האקטיבציה מסוג softmax משמשת לעתים קרובות כדי להשיג את ההסתברויות למחלקות. עבור משימות רגרסיה, נוירון בודד עם פונקציית אקטיבציה ליניארית עשוי להספיק.

עיבוד מוקדם של נתונים: עיבוד מוקדם של נתונים הוא קריטי. צריך לוודא שרצפי הקלט הם באורך אחיד, ולשקול טכניקות כמו ריפוד או חיתוך, או שרשור. בדרך כלל גם ננרמל או נתקן את הנתונים כך שיהיו עם ממוצע אפס ושונות בגודל יחידה, מה שיכול לעזור בתהליך ההתכנסות.

כוונון היפרפרמטרים: ניסוי עם היפרפרמטרים שונים כגון מספר השכבות הקונבולוציוניות, גדלי המסננים, מספר המסננים, קצב הלמידה, גודל הbatch ושיעורי הdropout. כאשר כוונון היפרפרמטר יכול להשפיע באופן משמעותי על ביצועי המודל.

התאמת יתר (overfitting): כמו כל רשת נוירונים, עבור CNN חד מימדית יכולים להתאים יתר על המידה את נתוני האימון. ניתן ליישם טכניקות רגולריזציה כמו dropout ורגולריזציית L2 כדי למנוע התאמת יתר.

הגדלת נתונים: כאשר עובדים עם נתונים מוגבלים, ניתן ליישם טכניקות של הגדלת נתונים כדי ליצור דוגמאות אימון נוספות. עבור נתוני סדרות במישור הזמן, זה עשוי לכלול הוספת רעש או עיוות זמן.

הערכת מודל: שימוש במדדי הערכה מתאימים (לדוגמה, דיוק או שגיאה ריבועית ממוצעת) ושימוש בטכניקות כמו cross validation כדי להעריך את ביצועי ההכללה של המודל.

Transfer learning: אם יש נתונים מוגבלים, נשקול להשתמש במודלים של CNN חד מימדיים מאומנים מראש, ולכוון אותם למשימה הספציפית שלנו. Transfer learning יכול לחסוך זמן אימון ולשפר תוצאות.

לסיכום, רשתות CNN חד מימדיות, הן כלים רבי עוצמה לעיבוד נתונים רציפים. הבנת הארכיטקטורה וקבלת בחירות מושכלות לגבי היפרפרמטרים ועיבוד מקדים של נתונים, חיוניים לבניית מודלים יעילים של אותן רשתות CNN. לעיתים קרובות נדרשים ניסויים וכוונון כדי להשיג את הביצועים הטובים ביותר במשימה הספציפית שנדרש אליה.

הכניסה לרשת

כעת נסביר כיצד קלטנו את קבצי השמע על מנת שנוכל להכניס אותם לרשת.

את קליטת המידע עשינו בכמה אופציות שונות, כאשר לאחר מכן נציג את התוצאות עבורן ונבחר כמובן באופציה הטובה ביותר.

ראשית קלטנו את קבצי השמע בתדר דגימה של $16{,}000[Hz]$, כאשר תדר זה הניב לנו בחלק ניכר של קבצי השמע $64{,}587$ דגימות. לכן כדי להגיע למספר אחיד של דגימות בכל אחד מהקבצים ריפדנו את כל הקבצים באפסים עד לגודל הזה. בנוסף כאופציה שנייה שרשרנו קובץ לעצמו עד שהגענו לגודל זה של $64{,}587$ דגימות לכל קובץ.

עשינו קליטה נוספת של קבצי השמע עם תדר דגימה גדול יותר-$32{,}000[Hz]$

כאן כבר קיבלנו מספר דגימות רב יותר $128{,}000$ דגימות. אנו מצפים שככל שתדר הדגימה יותר גדול, ובהתאם לכך מספר הדגימות, כך קבצי השמע יהיו יותר פשוטים לסיווג, נבדוק זאת במבחן התוצאה בהמשך.

עבור תדר הדגימה גדול יותר לא ריפדנו באפסים אלא רק שרשרנו כל קובץ לעצמו עד שהגענו לגודל של $128{,}000$ דגימות לכל קובץ וכך נוכל להתחיל להפעיל את הרשת.

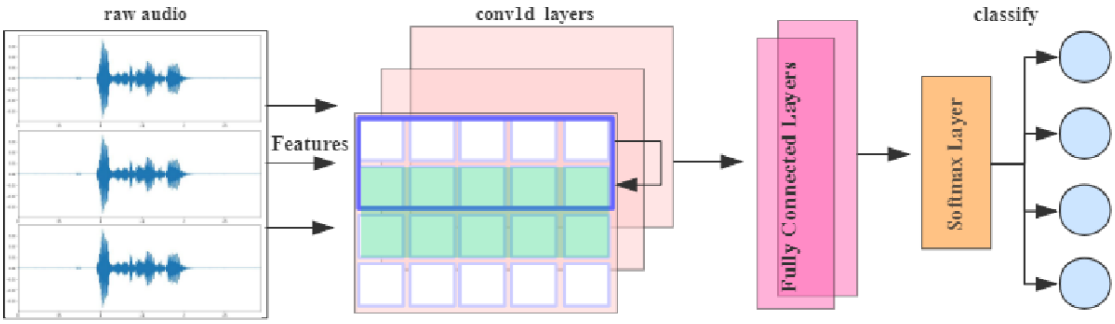
שכבות הרשת

כעת נפרט על הרשת עצמה, כפי שהפעלנו אותה בהתחלה. בהמשך ננסה לשפר את הרשת ע"י שינוי פרמטרים ועוד..

הרשת בנוייה משכבות קונבולוציה כמו שהסברנו מקודם, כאשר לאחר כל שכבה כזו, על מנת לשפר את תוצאות הרשת ביצענו נרמול של ערכי הנוירונים כך שהתוצאות יהיו יציבות יותר, ולאחר מכן הוספנו שכבת MaxPooling כדי שמימדי מוצא רשתות הקונבולוציה יקטנו והעומס החישובי ייקטן. לאחר 3 מבני שכבות הקונבולוציה שפרטנו, כדי להגיע למוצא של 10 קלאסים, השתמשנו בשכבות לינאריות שמצמצמות את המוצא ובעזרת פונקציית אקטיבציה של Relu הוצאנו את מוצא המערכת (בהמשך הוספנו שכבת Dropout כפי שמופיע בתמונה הנ"ל, נסביר שיקול זה בהמשך.)

```
class ConvNet(nn.Module):
    def __init__(self,p=0.5):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv1d(1,128,500,10)
        self.bn1 = nn.BatchNorm1d(128)
        self.pool1 = nn.MaxPool1d(2,4)
        self.conv2 = nn.Conv1d(64,32,3,1)
        self.bn2 = nn.BatchNorm1d(32)
        self.pool2 = nn.MaxPool1d(2,4)
        self.conv3 = nn.Conv1d(32,10,3,1)
        self.bn3 = nn.BatchNorm1d(10)
        self.pool3 = nn.MaxPool1d(2,4)
        self.drop_layer1 = nn.Dropout(p=p)
        self.fc1 = nn.Linear(380,256)
        self.drop_layer2 = nn.Dropout(p=p)
        self.fc2 = nn.Linear(256,64)
        self.drop_layer3 = nn.Dropout(p=p)
        self.fc3 = nn.Linear(64,10)
```

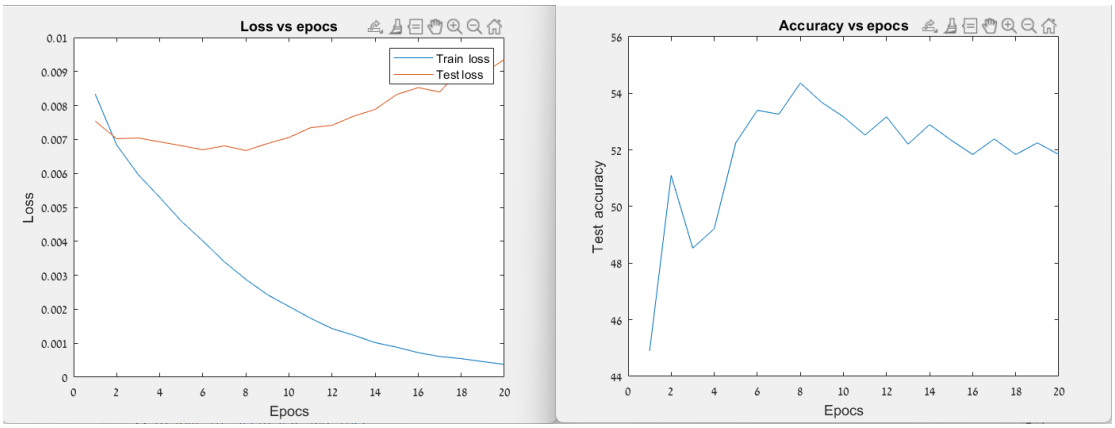
להלן דימוי של מבנה רשת הCNN החד מימדי.



ניתן לראות כי מקבלים קובץ אודיו, (בתחום הזמן בלבד), ולאחר מכן נעשה את הפעולות שהרחבנו עליהם קודם כדי להגיע לסיווג הטוב ביותר. (כמובן שנוסיף שכבות או תוספות אחרות שיעזרו לנו לשפר את התוצאות.

תוצאות

כעת נסביר על התוצאות. לאחר שהרצנו הרצה ראשונית, כאשר קלטנו את קבצי האודיו בתדירות דגימה של $16{,}000[Hz]$ וריפדנו את כל הקבצים באפסים כדי שיגיעו לגודל של הקובץ הארוך ביותר - $64{,}587$ דגימות, זאת כדי להיות בגודל אחיד של הכניסה לרשת. אלו התוצאות שקיבלנו:



ניתן לראות שפונקציית הloss מתחילה לעלות באזור הepoc השמיני, וגם אחוזי הדיוק מתחילים לרדת באותו רגע מהמקסימום של כ-55%. לכן ננסה לשנות את קליטת הקבצים כדי להגיע לתוצאות גבוהות ויציבות יותר. (במקרה זה הנתונים היו כמו בתמונה שצילמנו למעט שכבות הdropout שעוד לא הוספנו אותם.

השינוי שהחלטנו לעלות הוא במקום לרפד באפסים, לשרשר את הקבצים לעצמם עד אותו אורך מקסימלי של $64{,}587$ דגימות, זה בהנחה שהאפסים שהוספנו פגעו בתוצאות מכיוון שהאפסים מקשים על הסיווג כי הם לא מוסיפים שום מידע על אותו קובץ, ובעצם הדגימות האחרונות זהות בכל קבצי האודיו.

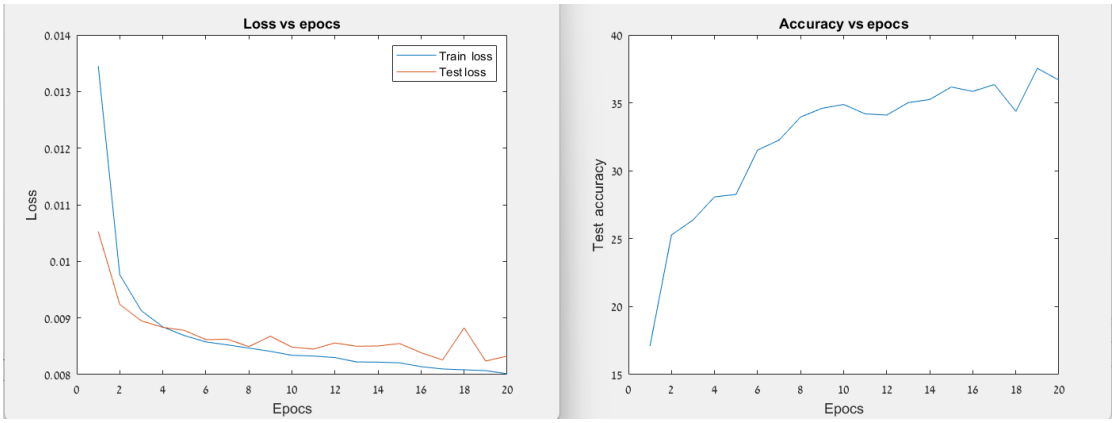
להלן הקוד שכתבנו כדי לבצע זאת:

```
if (len(audio_vec)<64587):
    x=(int)(64587/len(audio_vec))+1
    temp=audio_vec
    for i in range(x):
        temp = np.concatenate((temp,audio_vec))
    audio_vec=temp[1:64588]

if (len(audio_vec)>64587):
    audio_vec=audio_vec[1:64588]

return audio_vec, label
```

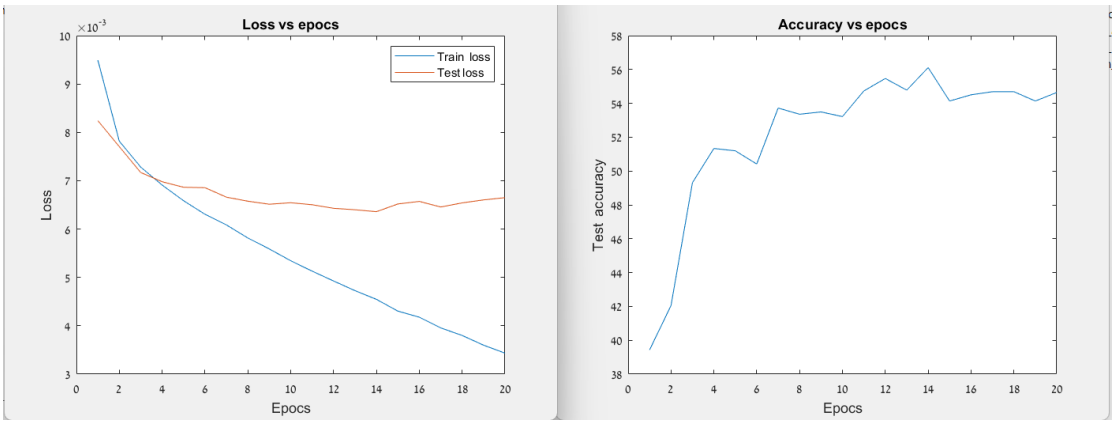
התוצאות עבור הרצה זו:



ניתן לראות כי פונקציית הloss ואחוזי הדיוק התייצבו במעט, אך עדיין התוצאות לא מספקות ולכן ננסה לשפר אף יותר את קליטת המידע.

הפעם הזו כבר שינינו את תדר הדגימה לתדר גבוה יותר - $32{,}000[Hz]$, וכתוצאה מכך מספר הדגימות גדלו, ולכן אנו מצפים לכך שיהיה יותר קל להבדיל בין קלאס אחד לשני.

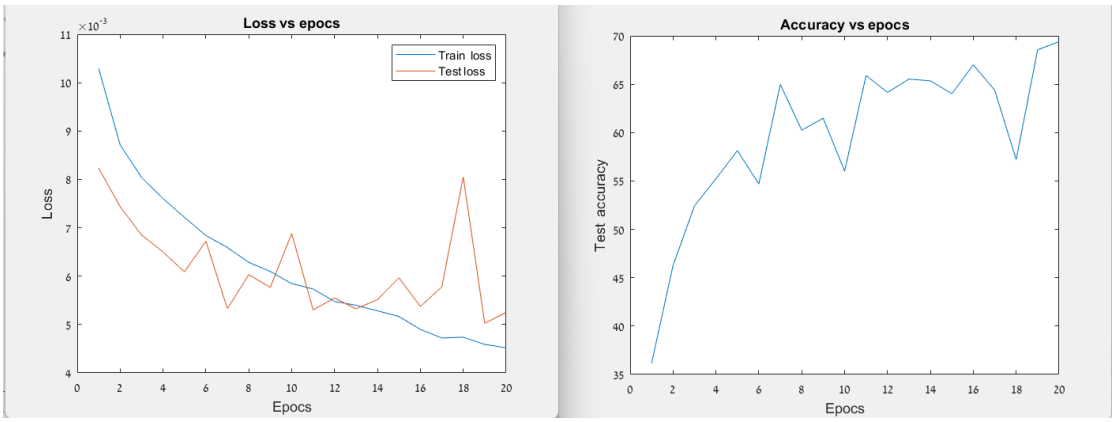
במקרה זה שרשרנו כל קובץ אודיו לעצמו כך שלאחר פעולה זו כל קבצי השמע באורך $128{,}000$ דגימות, להלן התוצאות:



כמו שציפינו התוצאות השתפרו, פונקציית הloss ירדה, ואחוזי הדיוק עלו לאזור ה – 56%, אך ניתן לראות שקיים overfitting גדול יחסית, והפרמטרים מותאמים יותר מדי לאימון ולא כלליים עבור כל כניסה.

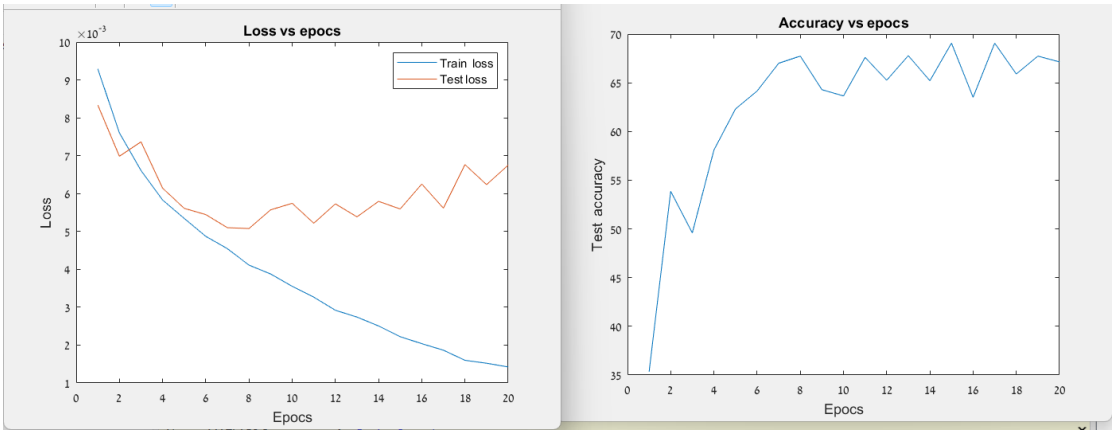
ננסה כעת באמצעות משחק עם היפרפרמטרים אחרים ברשת להגיע לתוצאות טובות יותר.

הוספנו רגולריזציה – $L2$, כך שהoverfitting יקטן ואולי התוצאות של הרשת ישתפרו בעקבות כך:



התוצאות אכן השתפרו במעט, אך הרשת נעשתה לא יציבה ועל כן ננסה לשנות אפשרויות אחרות.

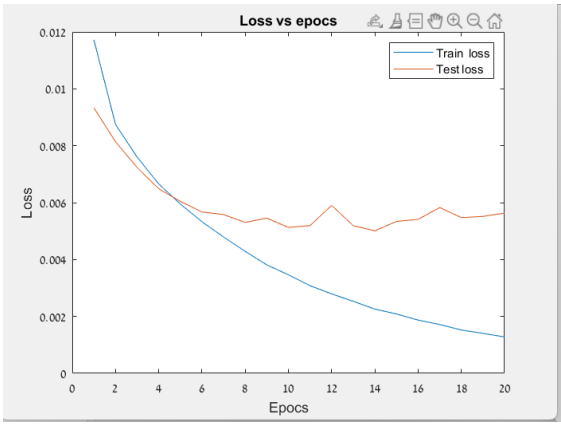
ניסינו לשנות את מימדי רשת הקונבולוציה, ואת גודל גרעין הקרנל. לאחר שינויים לא מעטים הגענו לתוצאות האלו:

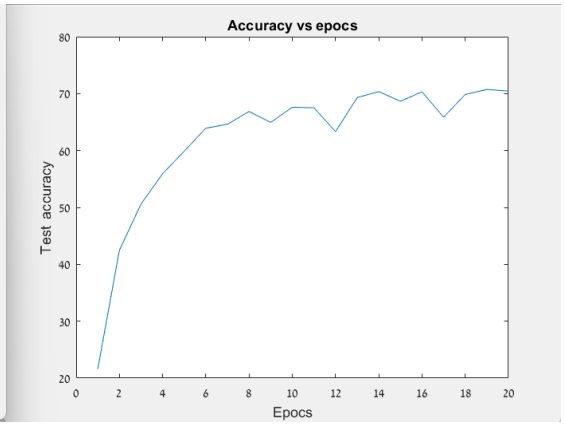


נראה שהתוצאות קצת יותר יציבות, אך עדיין אחוזי הדיוק לא מספיק גבוהים לטעמנו והoverfitting קיים.

כעת החלטנו להעמיק את מספר המסננים הקונבולציונים, והוספנו לאחר כל מסנן, מסנן נוסף (conv11, conv22, conv33):

```
super(ConvNet, self).__init__()
self.conv1 = nn.Conv1d(1,128,500,10)
self.conv11=nn.Conv1d(128,64,100,5)
self.bn1 = nn.BatchNorm1d(64)
self.pool1 = nn.MaxPool1d(2,4)
self.conv2 = nn.Conv1d(64,48,3,1)
self.conv22 = nn.Conv1d(48,32,3,1)
self.bn2 = nn.BatchNorm1d(32)
self.pool2 = nn.MaxPool1d(2,4)
self.conv3 = nn.Conv1d(32,20,3,1)
self.conv33 = nn.Conv1d(20,10,3,1)
self.bn3 = nn.BatchNorm1d(10)
self.pool3 = nn.MaxPool1d(2,4)
```



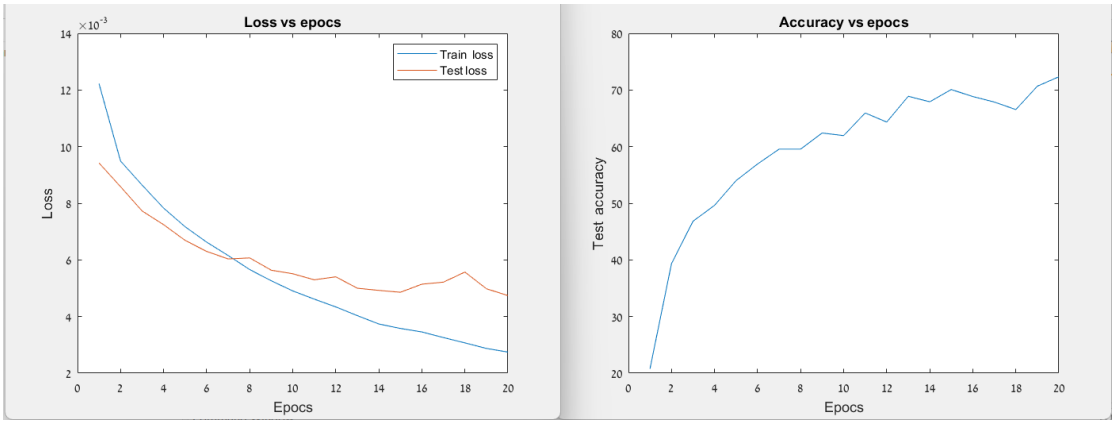


לאחר הוספנו זו ראינו שיפור בהתייצבות ואף לראשונה חצינו את ה70% דיוק.

לבסוף ניסינו להוסיף שכבות dropout בין השכבות הלינאריות, על מנת שזה יעזור לשפר את תוצאות הרשת:

```
class ConvNet(nn.Module):
    def __init__(self,p=0.2):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv1d(1,128,500,10
        self.conv11=nn.Conv1d(128,64,100,5)
        self.bn1 = nn.BatchNorm1d(64)
        self.pool1 = nn.MaxPool1d(2,4)
        self.conv2 = nn.Conv1d(64,48,3,1)
        self.conv22 = nn.Conv1d(48,32,3,1)
        self.bn2 = nn.BatchNorm1d(32)
        self.pool2 = nn.MaxPool1d(2,4)
        self.conv3 = nn.Conv1d(32,20,3,1)
        self.conv33 = nn.Conv1d(20,10,3,1)
        self.bn3 = nn.BatchNorm1d(10)
        self.pool3 = nn.MaxPool1d(2,4)
        self.drop_layer1 = nn.Dropout(p=p)
        self.fc1 = nn.Linear(380,256)
        self.drop_layer2 = nn.Dropout(p=p)
        self.fc2 = nn.Linear(256,64)
        self.drop_layer3 = nn.Dropout(p=p)
        self.fc3 = nn.Linear(64,10)
```

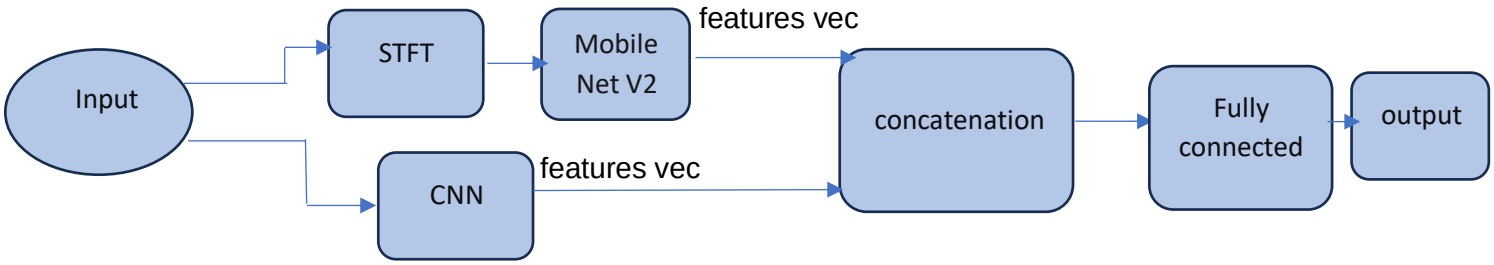
עבור dropout של 50% מהנוירונים קיבלנו ירידה דרסטית באחוזי הדיוק, אך עבור dropout של 20% קיבלנו תוצאות מעט יותר טובות ממקודם:



התוצאה הטובה ביותר אליה הגענו היא 73% דיוק, וגם הoverfitting קטן.

שילוב שתי הרשתות

בניית הרשת



פרמטרי הרשת

גם כאן ניסינו לאפטם את התוצאות של הרשת שלנו, והרצנו כמה הרצות שונות של ההיפר פרמטרים בקוד שלנו

כיוון שהdsi לא אפשר לנו להריץ $batch\ size$ מעל 16, נשארנו עם הפרמטרים:

1. $learning\ rate$
2. $optimization\ algorithm$

נציג בסוף את התוצאות לכל אחד מהערכים ונראה מה הגיע לתוצאות הטובות ביותר.

הכניסה לרשת

בחלק זה נרצה לשפר את התוצאות ע"י שרשור וקטור פיצ'רים של כל אחת מהרשתות.

לכן, הכניסה לרשת שלנו יהיה שילוב של שתי הכניסות משתי הרשתות הקודמות שהצגנו.

בנינו $class$ שמקבלת כקלט את ה$path$ של כל אחת מהרשתות- $STFT$ ווקטור של מידע, ומחזירה כפלט את הלייבל הנכון, מטריצת STFT ווקטור המידע של כל דוגמה. להלן הקוד:

```
class MyDataset_cat(Dataset):
  def __init__(self,path1,path2):
    self.path1=path1
    self.path2=path2
  def __len__(self):
    # Return the number of examples we have in the folder in the path
    os.chdir(self.path1)
    file_list = os.listdir()
    return len(file_list)
  def __getitem__(self,idx):
    new_path= self.path1+'train_data_'+str(idx)+'.pt'
    (image,label)=torch.load(new_path)
    image=image/abs(image).max()
    new_path= self.path2+'train_data_'+str(idx)+'.pt'
    (audio_vec,label)=torch.load(new_path)

    audio_vec=torch.tensor(audio_vec/abs(audio_vec).max())
    label=torch.nn.functional.one_hot(torch.tensor(label),num_classes=10)

    return (image,audio_vec,label)

path1 = '/root/data_set/dataset_original/'
path2 = '/root/data_set/dataset_audio_concatenate_32khz/'
dataset = MyDataset_cat(path1,path2)
```

שכבות הרשת

כמו שהסברנו, בחלק זה לקחנו את החלקים המרכזיים של שתי הרשתות הקודמות שלנו, ושרשנו אותם כדי להגיע לוקטור פיצ'רים מחובר.

לאחר שחיברנו, נעביר וקטור זה ברשת קטנה שמורכבת מ4 שכבות $fully\ conected$ עם פונקציית אקטיבציה $ReLU$, כאשר לכל אחד נוסיף שכבת $dropout$ עם הסתברות p משתנה., כדי למנוע $overfitting$.

הקוד שמימשנו:

```
class Network_cat(nn.Module):
    def __init__(self):
        super(Network_cat, self).__init__()
        #self.conv1 = nn.Conv1d(1,128,200,10)
        #self.conv11=nn.Conv1d(128,64,30,5)
        #self.bn1 = nn.BatchNorm1d(64)
        #self.pool1 = nn.MaxPool1d(2,4)
        self.drop_layer1 = nn.Dropout(p=0.2)
        self.fc2 = nn.Linear(1380, 512)
        self.drop_layer2 = nn.Dropout(p=0.2)
        self.fc3 = nn.Linear(512, 64)
        self.drop_layer3 = nn.Dropout(p=0.2)
        self.fc4 = nn.Linear(64, output_dim)


    def forward(self, x):

        x = self.drop_layer1(F.relu(x))
        x = self.drop_layer2(F.relu(self.fc2(x)))
        x = self.drop_layer3(F.relu(self.fc3(x)))
        x = self.fc4(x)

        return x

```

מוצא הרשת

כיוון שהרשת שלנו הינה רשת סיווג, מוצא הרשת הינו הלייבל המשוער של כל הקלטה.

תוצאות הרשת

תוצאות עבור רשת הסיווג

שלב אימון הרשת
כדי לאמן את הרשת עבדנו עם פונקציית מחיר $BCEWithLogitsLoss$.

הקוד שמימשנו:

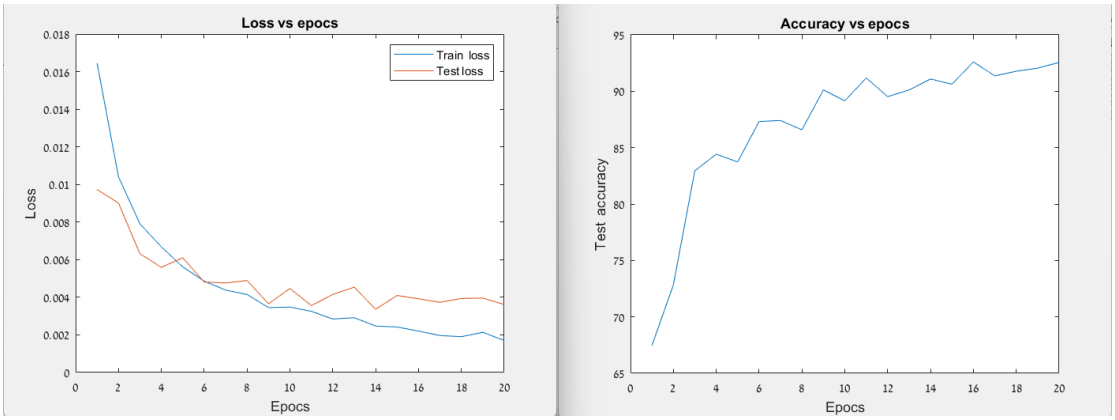
```
def train(n_epoch):
  batch_idx=0
  loss_all=0
  network_cat.train()
  network.train()
  network_CNN.train()
  for (data_image,data_vec,target1) in train_loader_MobileNet:
    data_image=data_image.to(device)
    data_vec=data_vec.to(device)
    target1=target1.to(device)
    optimizer_MobileNetV2.zero_grad()
    optimizer_cat.zero_grad()
    optimizer_CNN.zero_grad()
    output_Mobile_Net = network(data_image)
    output_CNN=network_CNN(data_vec)
    output_cat=torch.cat((output_Mobile_Net,output_CNN),dim=1)
    output_cat_final=network_cat(output_cat)
    loss = criterion(output_cat_final, target1.float())
    loss.backward()
    torch.nn.utils.clip_grad_norm_(network_cat.parameters(),1)
    torch.nn.utils.clip_grad_norm_(network.parameters(),1)
    torch.nn.utils.clip_grad_norm_(network_CNN.parameters(),1)
    optimizer_MobileNetV2.step()
    optimizer_cat.step()
    optimizer_CNN.step()
    if batch_idx % log_interval == 0:
      print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(n_epoch, batch_idx * len(data_image), len
      train_losses.append(loss.item())
      train_counter.append((batch_idx*64) + ((n_epoch-1)*len(train_loader_MobileNet.dataset)))
    loss_all+=loss.item() /len(train_loader_MobileNet.dataset)
    batch_idx=batch_idx+1
  return loss_all
```

כאמור בתמונה זו אנו רואים את כל השלבים שתיארנו בסכימה, כולל שלב החיבור במימד ה$feature$-ים.

איפטמנו כל משתנה בנפרד- קודם את ה$learning\ rate$ לאחר מכן את ה$droupout$, $batch\ size$ ולבסוף את $optimization\ algorithm$.

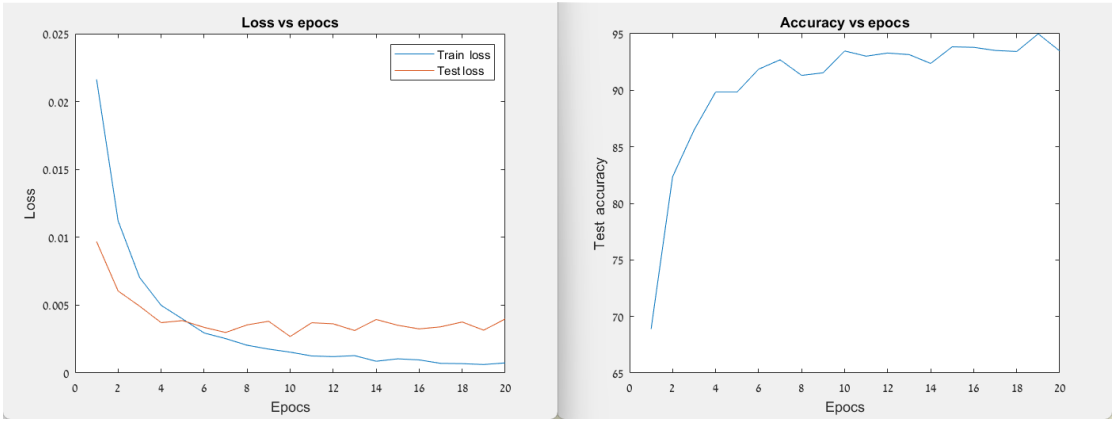
כעת נסביר על התוצאות.

ראשית הרצנו את עבור learning rate של 0.0005 ולהלן התוצאות:



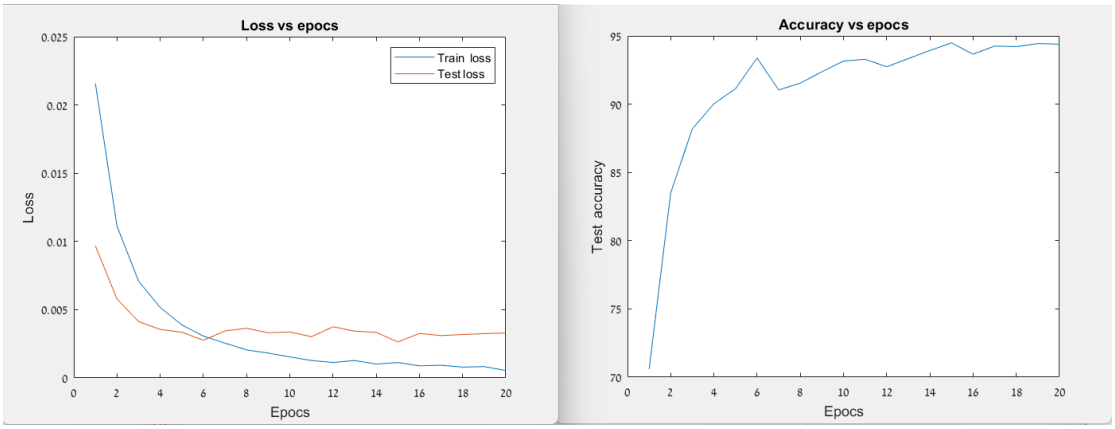
הגענו לתוצאות טובות יחסית, ננסה לשפר אף יותר את אחוזי הדיוק מ92%.

ניסינו לשנות את הlearning rate במעט ולשפר את הדיוק:



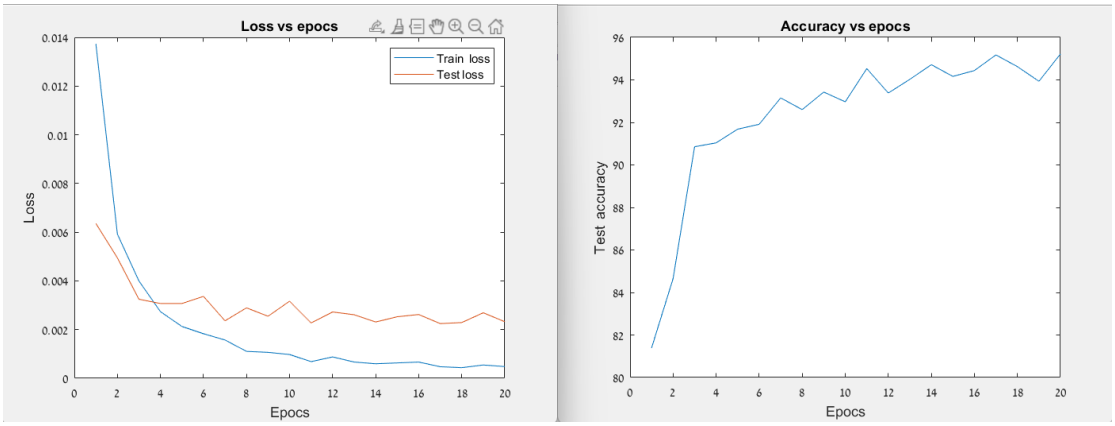
הצלחנו לשפר את האחוזים ל94 אחוזי דיוק.

ננסה להוסיף שכבת dropout על מנת לשפר אף יותר את התוצאות. בחרנו את ההסתברות של הdropout ל0.5:



נראה שהתוצאות התייצבו במעט אך האחוזים לא השתפרו.

ננסה כעת לשנות את ההסתברות לdropout ל0.2 , להלן התוצאות:



הצלחנו לשפר את אחוזי הדיוק ל95% , ובכך הגענו לאחוזים הגבוהים ביותר שראינו עד כה מ2 הרשתות.

מסקנות וסיכום

בפרויקט עבדנו בשלושה שלבים עיקריים:

א) בנינו רשת סיווג המבוססת על תמונות שיצרנו באמצעות התמרת STFT , וכך הסיווג היה גם במימד התדר.
עבור רשת זו המבוססת על רשת מוכרת - MobileNetV2, קיבלנו תוצאות יחסית טובות של כ- 94%.
לפי הבנתנו, בעקבות קבלת המידע גם במימד התדר, הצלחנו לחלץ יותר מידע שלבסוף עזר לנו לסווג בהצלחה גדולה יותר את המידע.

ב) בנינו את הרשת סיווג השנייה, שקולטת את המידע בצורת קבצי אודיו, כלומר מימד הזמן בלבד. ובעזרת רשת קונבולוציה של נוירונים חד מימדית, סיווגנו את המידע עם הצלחה מוגבלת – 72% בלבד.

ג) לבסוף שרשרנו את המוצאים של שתי הרשתות, ובעצם ע"י כך היה בידינו מידע מ2 הרשתות. לאחר שיפורים הצלחנו להגיע לתוצאה הטובה של 96% הצלחה.

בעצם השתמשנו בידע שרכשנו בקורס "עיבוד ספרתי של אותות 2" בעיקר בשביל הרשת הראשונה, וזאת כדי להגיע לכניסות בדמות תמונות שנוצרו באמצעות התמרת STFT.
בנוסף השתמשנו בידע מהקורס "למידה עמוקה" עבור כל אחת מהרשתות הנ"ל, קודם כל על מנת ליצור את רשתות הנוירונים, ושנית עבור שיפור הרשתות באמצעות כל השיטות שהזכרנו במהלך הפרויקט.

נסכם כי הפרויקט היה מאוד מהנה ומאתגר עבורנו. למדנו המון מגורמים שונים, בין מהאינטרנט באמצעות אתרים שהיו רלוונטיים לפרויקט, ובין אם ממנחת הפרויקט שלנו – דניאל לוי, שבכל פעם מחדש נענתה לבקשתנו ועזרה לנו בשמחה רבה בקשיים שהיו לנו במהלך הפרויקט. אז ננצל את הבמה הזו כדי להגיד לה תודה רבה על כל העזרה במהלך השנה, זה בכלל לא מובן מאליו. בנוסף נרצה להודות גם לפרופ' שרון גנות ולפרופ' יעקב גולדברגר, שמהם למדנו את הקורסים הרלוונטים לפרויקט, וחלק גדול מהידע שצברנו ועזר לנו ביישום הפרויקט זה בזכותם.

רעיונות להמשך שיפור הפרויקט

נוסף לתחומי זמן ותדר, ישנם תחומים או ייצוגים אחרים שאפשר לחקור לסיווג נתוני אודיו. להלן מספר תחומים וגישות חלופיות:

כרומגרמה - כרומגרמות מייצגות את תכולת האנרגיה של מחלקות גובה שונות (תווים) באות שמע. הם משמשים לעתים קרובות למשימות סיווג מוזיקה, כגון סיווג ז'אנר.

ניגודיות ספקטרלית - ניגודיות ספקטרלית היא תכונה המודדת את ההבדל באמפליטודה בין פסגות ועמקים בספקטרום האודיו.

ייצוגים ספציפיים לתחום - בהתאם לאופי משימת זיהוי דפוסי השמע, ייתכן שנגלה ייצוגים ספציפיים לתחום הרלוונטיים במיוחד. לדוגמה, בתחום סיווג הסאונד הסביבתי, ישנם מאפיינים אקוסטיים מיוחדים כמו עוצמה אקוסטית ותארי אירועי סאונד.

הטבעות ממודלים מאומנים מראש: אפשר להשתמש במודלים של למידה עמוקה שנלמדו מראש (למשל, VGGish, YAMNet) כדי לחלץ הטבעות מנתוני האודיו שלנו. לאחר מכן ניתן להשתמש בהטבעות אלו כקלט למודל הסיווג.

לסיכום, בחירת התחום או הייצוג תלויה באופי נתוני האודיו ובדפוסים הספציפיים שברצוננו לזהות. לעתים קרובות מומלץ להתנסות עם ייצוגים שונים ולטכניקות חילוץ כדי לראות איזו מהן עובדת הכי טוב עבור משימת הסיווג הספציפית שלנו. במקרים מסוימים, שילוב של ייצוגים מרובים או שימוש במודלים היברידיים יכולים להוביל לשיפור הביצועים.

ביבליוגרפיה

1)

המאמר עליו מבוסס הפרויקט:

PANNs_Large-Scale_Pretrained_Audio_Neural_Networks_for_Audio_Pattern_Recognition

2)

https://eureka.org.il/item/44717/%D7%9E%D7%94%D7%9F-%D7%A8%D7%A9%D7%AA%D7%95%D7%AA-%D7%A0%D7%95%D7%99%D7%A8%D7%95%D7%A0%D7%99%D7%9D-%D7%9E%D7%9E%D7%95%D7%97%D7%A9%D7%91%D7%95%D7%AA

3) תרגולים מהקורס "עיבוד ספרתי של אותות 2"

4)

https://www.youtube.com/watch?v=88FFnqt5MNI