CAN KORKUT

141180046

BM400

STAJ RAPORU

SESTEK – Ses ve İletişim Bilgisayar Teknolojileri
Ankara Ofisi

İÇİNDEKİLER

Sayfa

İÇİNDEKİLER i

ÇİZELGELERİN LİSTESİ iii

ŞEKİLLERİN LİSTESİ iv

SİMGELER VE KISALTMALAR v

1. GİRİŞ 1

2. FİRMA HAKKINDA BİLGİ 3

2.1. Kuruluşun Adı 3

2.2. Kuruluşun Yeri 3

2.3. Kuruluşta Çalışanlar Hakkında Bilgi 3

2.4. Kuruluş Faaliyet Alanı 3

2.5. Kuruluşun Kısa Tarihçesi 3

3. STAJ RAPORU 4

3.1. Gerekli Kurulumlar 4

3.1.1. Jupyter Notebook Kurulumu 4

3.1.2. CUDA Kurulumu 4

3.1.1. PyTorch Kurulumu 5

3.2. Problemin Tanımı ve Literatür Taraması 5

3.2.1. Ses Verisinden Öz Nitelik Çıkarımı 5

3.2.2. Yapay Sinir Ağı Modelinin Tasarlanması 7

3.3.Evrişimsel Sinir Ağları 9

3.4. Uzun Kısa Süreli Modeli (LSTM) 11

3.5. Eğitim Sonuçlarının Değerlendirilmesi 12

3.6. DLL Dosyasının Oluşturulması 14

3.7. Kullanıcı Arayüzü 16

4. SONUÇ 18

KAYNAKLAR 19

EKLER 21

EK-1. Staj başarı belgesi bölüm kopyası 21

EK-2. Kaynak Kodlar 22

ÇİZELGELERİN LİSTESİ

Çizelge **Sayfa**

Çizelge 3.1. Eğitim Sonuçları…….14

ŞEKİLLERİN LİSTESİ

Şekil **Sayfa**

Şekil 3.1: Analog Sinyalin Dijitale Dönüşmesi 6

Şekil 3.2: Ses Sinyalinden MFCC Çıkarımı 6

Şekil 3.3: Ses Sinyalinden Spektrogram 7

Şekil 3.4: Biyolojik Sinir Hücresi 8

Şekil 3.5: Perceptron Modeli 8

Şekil 3.6: Eğitim Sonuçları 9

Şekil 3.7: Konvolüsyonel Yapay Sinir Ağı Mimarisi. 10

Şekil 3.8: Evrişimsel Sinir Ağı 11

Şekil 3.9: CNN Model Sonuçları. 11

Şekil 3.10: LSTM Hücresi 12

Şekil 3.11: LSTM Model Sonuçları. 13

Şekil 3.12: Libtorch Kütüphanesinin indirilmesi 15

Şekil 3.13: Örnek DLL Söz Dizimi 16

Simgeler ve Kısaltmalar

Bu çalışmada kullanılmış simgeler ve kısaltmalar, açıklamaları ile birlikte aşağıda sunulmuştur.

Simgeler	**Açıklamalar**
mV	miliVolt
Na	Sodyum

Kısaltmalar	**Açıklamalar**
CNN	Convolutional Neural Network
MFCC	Mel Frequency Cepstral Coefficient
LSTM	Long Short Term Memory
i-vector	identification vector

1. GİRİŞ

Bulunduğumuz yıllarda telefon görüşmeleri ve sesli iletişim hayatımızın pek çok alanında yaygın olarak kullanılmaktadır. Sesli iletişim gerçekleştirilirken konuşmacı ile görsel temas her zaman için mümkün olmamaktadır. Bu durum ikili iletişimdeki etkileşimi verimsiz kılmaktadır. Özellikle bankacılık, sigortacılık gibi duygusal etkileşimin önemli olduğu alanlarda bu problem daha çok ön plana çıkmaktadır. Staj süresince ele alınan problem çerçevesinde bahsedilen probleme çözüm sunacak, konuşmacı duygu analizi başlıklı proje geliştirilmiştir. Geliştirilen proje ile konuşmacıdan elde edilen ses verisi işlenerek, konuşmacının sinirli olup olmadığının saptanmasını kapsamaktadır. Böylelikle sesli iletişimin kullanıldığı ortamlarda duygu analizi gerçekleştirilerek, iletişimden alınan verim artırılabilir. Bahsedilen projenin gerçekleştirilebilmesi için ağırlıklı olarak makine öğrenmesi ve yapay sinir ağları kullanılmaktadır. Bu kapsamda yapılan çalışmalar ses ve konuşma sınıflandırma üst başlığında değerlendirilebilir. Ses sınıflandırma alanında yapılan çalışmalarda kullanılan yöntemler ses verisini ön işleme, işlenmiş veriden öznitelik elde etme, elde edilen öz nitelikleri makine öğrenmesi modellerinde eğitme aşamalarından oluşmaktadır. Araştırılan benzer projelerde ise ses verisini spektrogram resmine dönüştürerek, sınıflandırma işleminin evrişimsel sinir ağlarında yapılmasına dayanmaktadır [1]. Bu çalışmalara ek olarak 2011 yılında yayınlanan konuşma tanıma aracı Kaldi [2], ses ve konuşma tanıma, sınıflandırma projelerinde yaygın olarak kullanılmaktadır. Staj süresinde de Kaldi aracından faydalanılmıştır.

2. FİRMA HAKKINDA BİLGİ

Sestek diyaloğa dayalı yapay zekâ ve analiz çözümleri geliştiren global bir teknoloji firmasıdır. 2000 senesinden beri bu alanda faaliyet gösteren Sestek, kurumların veri odaklı karar vermelerine, verimli çalışmalarına ve daha iyi müşteri deneyimi sunmalarına yönelik çözümler sunar. Sestek, Ar-Ge merkezi olarak geliştirdiği yüksek teknoloji çözümlerinin yanı sıra; üniversite-sanayi iş birliğindeki lider rolüyle öne çıkmaktadır. Sestek'in sunduğu yapay zekâ destekli çözümler, konuşma tanıma, konuşma sentezi, doğal dil işleme ve ses biometrisi gibi teknolojileri kullanmaktadır.

Sestek, İstanbul ve Ankara'da olmak üzere iki Ar-Ge merkezinin yanı sıra Dubai'de bulunan satış ofisi ile faaliyetlerine devam etmektedir. 70'i Ar-Ge mühendisi olan 90 kişilik ekibi ile 24 devlet destekli Ar-Ge projesi geliştirmiş Sestek'in diyaloğa dayalı çözümler üzerine 11 patenti mevcuttur. Gelirinin %40'ını Ar-Ge faaliyetlerine ayıran Sestek, CEO'su Prof. Levent Arslan önderliğinde proje çalışmalarını yüksek oranda ticari faydaya çevirmektedir.

2.1 Kurulun Adı

Sestek Ses ve İletişim Bilgisayar Teknolojileri Anonim Şirketi.

2.2 Kuruluşun Yeri

Adres: Ayazağa Mahallesi Cendere Cad. 109B Vadistanbul Bulvar 1B Blok Ofis No:4 / 34396 Sarıyer, İstanbul – Türkiye

ARGE Merkezi: Mustafa Kemal Mahallesi Dumlupınar Bulvarı 274/7 Mahall Ankara B Blok No:178 Çankaya/Ankara

2.3 Kuruluşta Çalışanlar Hakkında Bilgi

Sestek firması İstanbul, Ankara ve Dubai'de faaliyet göstermektedir. Toplamda 90 kişilik çalışanı bulunmaktadır. 90 kişilik ekibin 70 kişisi ise Ar – Ge mühendisi ünvanı ile çalışmaktadır. Bu kapsamda firma gelirinin %40'ını Ar – Ge faaliyetlerine ayırmaktadır.

2.4 Kuruluşun Faaliyet Alanı

Sestek, Ar-Ge merkezi olarak geliştirdiği yüksek teknoloji çözümlerinin yanı sıra; üniversite-sanayi iş birliğindeki lider rolüyle öne çıkmaktadır. Sestek'in sunduğu yapay zekâ destekli çözümler, konuşma tanıma, konuşma sentezi, doğal dil işleme ve ses biometrisi gibi teknolojileri kullanmaktadır.

2.5 Kuruluşun Kısa Tarihçesi

Sestek'in temelini, Levent Arslan'ın Boğaziçi Üniversitesi'nde elektronik mühendisliği öğrenimi gördükten sonra Duke Üniversitesi'nde yaptığı "sesten gürültü temizleme" konusundaki mastır ve aksan alanındaki doktora çalışmaları oluşturuyor. Doktora programının son iki yazında Texas Instruments'ta çalışırken gürültü temizleme ve konuşma tanımayla ilgili üç tane patent alan Levent Arslan ardından, Washington'da ses konusunda araştırma yapan Entropic Research şirketinde görev yaparken bir kişinin sesini bilgisayar yardımıyla başka birinin sesine dönüştüren sistemi üretti. Bu, alanında başarıya ulaşan ilk sistem oldu ve Arslan bunun da patentini aldı. Entropic Research daha sonra Microsoft tarafından satın alındı. İki yıl Entropic Research'de çalıştıktan sonra Türkiye'ye dönen Arslan, Koç Grubu ile birlikte GVZ şirketini kurdu. Paralel olarak ABD'deki ses dönüştürmeyle ilgili çalışmalar için de Sestek'i. İki şirket de 2000'de kuruldu, 2007'de ise Prof. Dr. Levent Arslan, GVZ'deki Koç Holding hisselerini geri aldı.

Kurulduğundan bu yana şirketin geliştirdiği başlıca ürünleri Konuşma Sentezi, Konuşma Tanıma, Konuşma Analizi, Sesli İmza, Doğal Diyalog ve en son da Chatbot oluşturuyor. Bu ürünler insan sesini yazıya, yazıyı insan sesine çeviriyor, ses değişkenlerini çözümlüyor, ses ile kişiyi ilişkilendiriyor, kimliği belirliyor ve kanıtlıyor. Arslan bu temel işlevleri çeşitlendirip aynı teknolojiler üzerinden farklı ihtiyaçları karşılayan ürünler de geliştiriyor. Sestek şimdi özellikle Nuance'ın güçlü oldubir alana iddialı bir ürünle giriyor: Chatbot. Doğal konuşma üslubu içinde insan-makine iletişimini sağlayacak bu ürün IVR, Web sayfası ya da mobil uygulamadan giriş olanağı veriyor ve ortak bir platform oluşturarak firmaların müşterileriyle iletişimi için yazılı ve sözlü pek çok kanal sunuyor. Yakında sonuçları alınacak ilk uygulamalar ise Chatbot'un dengeleri ne yönde değiştireceğini gösterecek.

3 STAJ RAPORU

Sestek firması ses tanıma, sesi yazı dönüştürme, yazıyı sese dönüştürme gibi ses teknolojilerinde çözüm sunmaktadır. Bu doğrultuda staj süresince ses analizi üzerine çalışmalar gerçekleştirdim. İlk olarak çalışmaları gerçekleştireceğim platformun kurulumu ile ilgilendim. Staj süresince kullanacağım PyTorch, Sklearn, Python gibi kütüphaneleri yükledim. Bu kütüphanelerin kurulum aşaması aşağıda verilmiştir.

3.2 Gerekli Kurulumları

Bu bölümde staj süresince kullanılacak kütüphane ve ortamların kurulumları anlatılmıştır. Staj süresince ağırlıklı olarak makine öğrenmesi uygulamaları gerçekleştirileceği için bu alanda yaygın olarak kullanılan anaconda geliştirme aracı kullanılmıştır. Anaconda geliştirme ortamı ile 1000'den fazla bilimsel hesaplama kütüphanesini barındırır. Bunun için öncelikle anaconda aracı kurulmuştur. Geri kalan kurulumlar bu araç üzerinden gerçekleştirilmiştir.

3.1.1 Jupyter Notebook Kurulumu

Jupyter (eski adıyla IPython Notebook) tarayıcı içinde kod çalıştırmanızı, grafikler çizmenizi, yazı ve resim eklemenizi sağlayan oldukça popüler bir araçtır. Anaconda üzerinden "conda install jupyter "komutu ile kolaylıkla kurulmaktadır. Staj süresince geliştirme jupyter notebook üzerinden gerçekleştirilmiştir.

3.1.2. CUDA Kurulumu

Grafik İşlem Birimi (GPU)'nin temel görevi bilgisayarda oluşturulan görüntülerin ekrana verilmesini sağlamaktır. Bu yüzden ilk GPU'lar sadece bu görevi yerine getirmekteydi. Zaman içerisinde Merkezi İşlem Birimi (CPU)'nin karşılaşılan büyük hesaplama problemlerinde yetersiz kaması üzerine GPU'nun donanımsal paralelliğinden yararlanma fikri ortaya çıkmıştır. GPU'ların programlanabilir bir arayüze sahip olmasını ve yüksek seviyeli dillerle programlanabilmesini sağlamak için GPGPU modeli oluşturulmuştur. CUDA (Compute Unified Device Architecture), NVIDIA firmasının 2006 yılında GPU'nun donanımsal hesaplama gücünden faydalanmak amacıyla sunduğu paralel hesaplama mimarisidir. Linux, Windows ve Mac Osx platformları üzerinde çalışabilmektedir. FORTRAN, C/C++ ve Python gibi dilleri destekleyen bir API'dir. Rakiplerine göre avantajları paylaşımlı bellek kullanımı, GPU'dan daha hızlı veri okuma ve bit düzeyinde işlem yapılabilmesine olanak sağlaması olarak sayılabilir.

3.1.3 PyTorch Kurulumu

PyTorch, bilgisayar görüşü ve doğal dil işleme gibi uygulamalar için kullanılan Torch kütüphanesine dayanan açık kaynaklı bir makine öğrenme kütüphanesidir. Öncelikle Facebook'un yapay zekâ araştırma grubu tarafından geliştirilmiştir. Staj süresince bu kütüphane kullanılmıştır. Kütüphane kurulumu diğer derin öğrenme kütüphanelerine nazaran daha kolaydır. Anaconda üzerinden " conda install pytorch torchvision -c soumith " komutu ile kurulabilir.

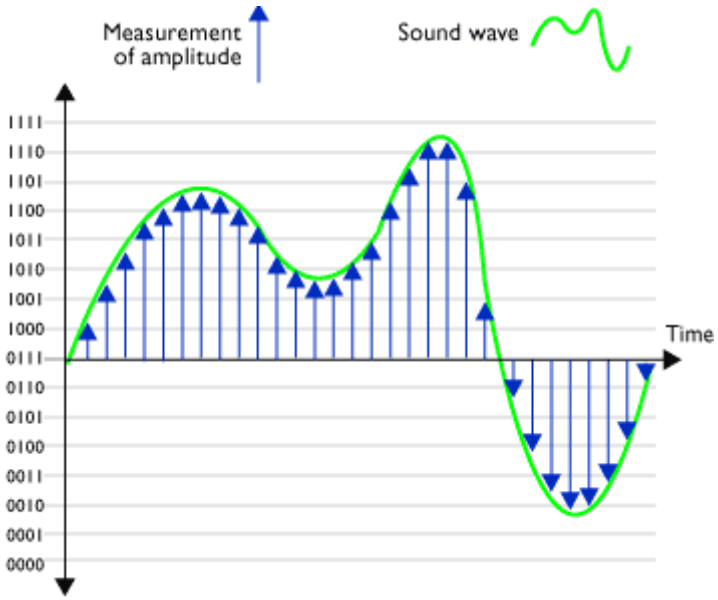
Yukarıda belirtilen aşamalar sırası ile gerçekleştirilerek staj süresince kullanılacak teknolojilerin kurulumu tamamlanmıştır.

3.2 Problemin Tanımı ve Literatür Taraması

Staj süresince incelenecek problem, konuşmacıdan alınan ses verisinin sınıflandırılması temeline dayanmaktadır. Yapılan çalışmalarda belirli sayıda sınıflandırılmış ses verisi işlenerek, yeni gelen ses verisinin hangi sınıfa ait olabileceği tahmin edilmektedir. Ses verisinin modelde eğitilmesi bazı temel adımları içermektedir. Bu adımlar aşağıda açıklamaları ile birlikte verilmiştir.

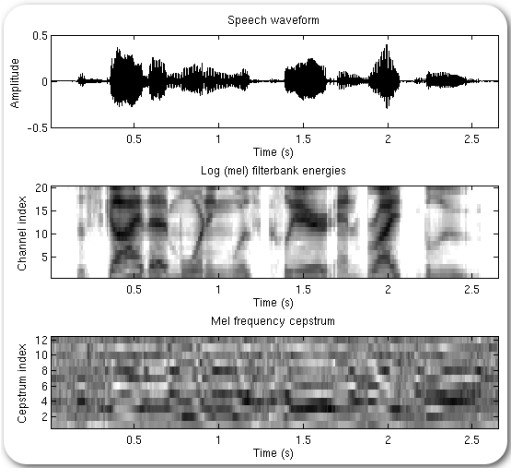
3.2.1 Ses Verisinden Öz Nitelik Elde Etme

Konuşma sinyalinin, bilgisayarlar ve örüntü tanıma algoritmaları için kullanılabilir olması için dijital örneklere dönüştürülmesi gerekmektedir. Analog ortamdaki sinyal, mikrofon vasıtası ve ses kartı ile dijital örneklere aktarılmaktadır. Bu aktarım sürecinde, analog bir sinyalinin belli zaman aralıklarında belli sayıda örnekleri alınıp dijital sayılar şeklinde kaydediliyor. Dijitalleştirme sonucunda, analog ortamdaki sürekli olan bir sinyal dijital ortamda kesikli sinyal örneklerine dönüştürülüyor. Bu örnekleme süreci iki farklı parametre ile ifade edilip birincisi örnekleme sıklığı ikincisi de örneklerin tutulmasında kullanılan bit sayısıdır. Bu şekildeki s(t) fonksiyonu örneklenmesi istenilen sürekli bir sinyali gösterip ve örnekleme işlemi bu fonksiyonun T zaman aralığındaki değerleridir. Ayrıca örnekleme sıklığı fs ile gösterilip, bir saniye içerisinde s(t) fonksiyonundan alınan örnek sayısını ifade etmektedir.



Şekil 3.1: Analog Sinyalin Dijitale Dönüşmesi [3]

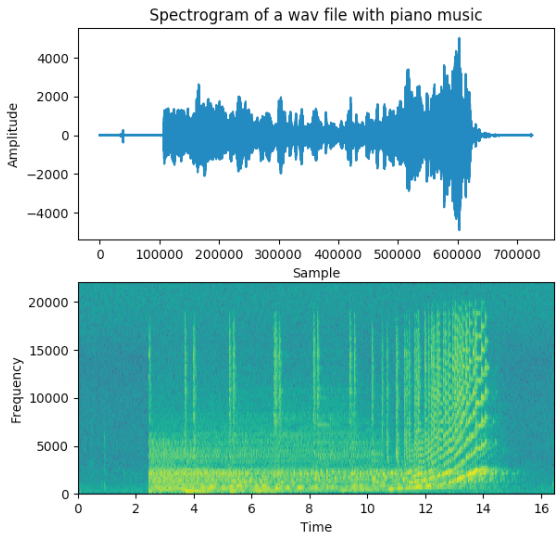
Dijitale dönüştürülen sinyalden öz nitelik çıkarmak için uygulanan en yaygın yöntemlerden biri Mel Frequncy Cepstral Coefficients (MFCC) yöntemidir. Mel-frekans hücresi, bir lineer olmayan mel frekans skalasındaki bir log güç spektrumunun lineer bir kosinüs dönüşümüne dayanan bir sesin kısa süreli güç spektrumunun bir temsilidir. Mel frekanslı cepstral katsayıları, toplu olarak bir MFC'yi oluşturan katsayılardır. Literatürde ses sintalinden mfcc çıkarımı için Kaldi aracı kullanılır.



Şekil 3.2: Ses Sinyalinden MFCC Çıkarımı [5]

Yukarıda anlatılan MFCC özniteliğinden farklı olarak son zamanlarda yapılan ses sınıflandırma çalışmalarında spectrogram resminin evrişimsel sinir ağları ile kullanıldığı

görülmektedir. İncelenen bazı çalışmalarda [4] farklı sınıfları içeren ses verisinin sırası ile dalga formatına daha sonra ise spektrograma dönüştürülerek öz niteliğin çıkarıldığı görülmektedir. Sonraki aşamada ise evrişimsel sinir ağı kullanılarak eğitim işlemi gerçekleştirilmiştir.



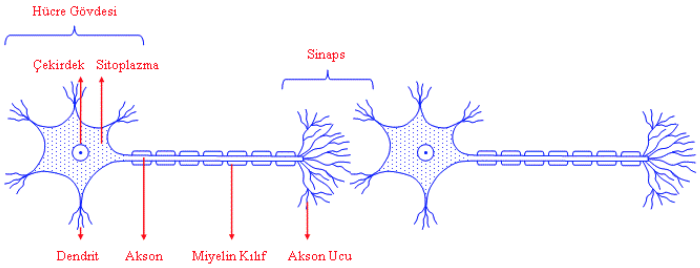
Şekil 3.3: Ses Sinyalinden Spektrogram Elde Edilmesi

Çalışma boyunca yukarıda anlatılanlara ek olarak Kaldi aracından elde edilen, konuşma tanıma uygulamalarında yaygın olarak kullanılan i-vector [5] öz niteliği kullanılmıştır. İlk aşamada sinirli ve normal olmak üzere iki farklı sınıf oluşturulmuştur. Daha sonra bu sınıflar altında ilgili sınıflara ait ses dosyaları toplanmıştır. Ardından yukarıda bahsedilen öz nitelik çıkarma işlemleri gerçekleştirilmiştir. Öz nitelik çıkarım aşamasının tamamlanması ile birlikte bir sonraki aşama olan model tanımlama aşamasına geçilmiştir.

3.2.2 Yapay Sinir Ağı Modellerinin Tasarlanması

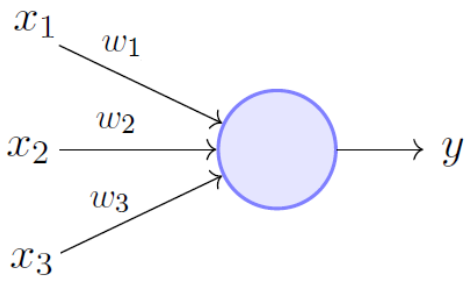
Yapay zekanın nihai amacı insan kadar başarılı tahmin, çıkarım ve sınıflandırma yapabilmektir. Bu doğrultuda bilim adamları insan beyninden yola çıkarak bir algoritma geliştirme uğraşına girişmişlerdir. İnsan beyni nöron adı verilen sinir hücrelerinden oluşmaktadır. Bu sinir hücrelerinden milyarlarca bulunmaktadır. Bir sinir hücresi ise akson ve dentrit adı ve verilen iki temel yapıdan meydana gelmektedir. Akson darbelerin üretildiği bölüm, dentrit ise iletimin gerçekleştirildiği bölümdür. Aksonda taşınan işaret sinapslara kimyasal taşıyıcılar yardımıyla iletilmektedir. Stoplazma -85mV ile polarizedir. -40mV (Na+ içeri): uyarma (+) akıma yol açar. -90mV (K+ dışarı): bastırma (-) akıma yol açar. Yani

belirli bir eşik gerilim değerinin üstünde iken hücre uyarılırken, diğer durumlarda hücre bastırılır. Σ Bu duruma göre çıkış işareti üretilmesine sinirsel hesaplama denir.



Şekil 3.4: Biyolojik Sinir Hücresi [7]

Yukarıda biyolojik canlılar tarafından kullanılan sinir hücresi anlatılmıştır. Biyolojik sinir ağından yola çıkarak bilim insanları tarafından yapay sinir ağı modellenmiştir. Yapay sinir ağları, insan beyninin bilgi işleme tekniğinden esinlenerek geliştirilmiş bir bilgiişlem teknolojisidir. YSA ile basit biyolojik sinir sisteminin çalışma şekli taklit edilir. Taklit edilen sinir hücreleri nöronlar içerirler ve bu nöronlar çeşitli şekillerde birbirlerine bağlanarak ağı oluştururlar. Günümüzde kullanılan yapay sinir ağı ilk defa 1957 yılında Frank Rosenblatt tarafından tanımlanmıştır. Tasarlanan yapay sinir ağının en küçük parçası (perceptron) temel olarak ağırlık kat sayısı ve işlemin gerçekleştiği hücreden oluşmaktadır.

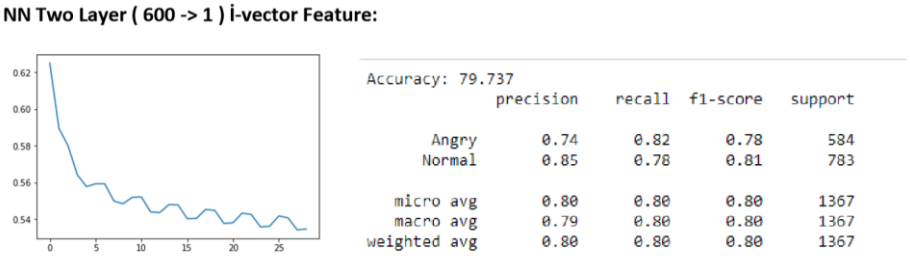


Şekil 3.5: Perceptron Modeli [8]

Yukarıdaki şekilde perceptron modeli verilmiştir. Bu modele göre x1, x2, x3 girdileri hücreye girdi olarak gelmektedir. W1, w2, w3 katsayıları ağırlıkları temsil etmektedir.

Yapılan bu işlem sonucu y çıktısı üretilmektedir. Verilen bu hücre birbiri ardına bağlanması ile yapay bir sinir ağı oluşturulur. Staj süresince ilk denemeler bu şekilde oluşturulan ağ modelinde gerçekleştirilmiştir.

Öz nitelik çıkarımı başlığında anlatıldığı üzere ilk etapta ses dosyalarından i-vector çıkarımı gerçekleştirilmiştir. Her ses dosyasına karşılık 600 adet i-vector dosyası elde edilmiştir. Bu vektörler ses dosyalarına karşılık gelecek dosyalarda tutulmuştur. Sonra ise 600 adet girdiye sahip yapay sinir ağı modeli PyTorch kütüphanesi ile tasarlanmıştır. Tasarımın gerçekleştirilmesinden sonra ise vektörler modele verilerek eğitim gerçekleştirilmiştir. Elde edilen eğitim sonuçları aşağıdaki gibidir.

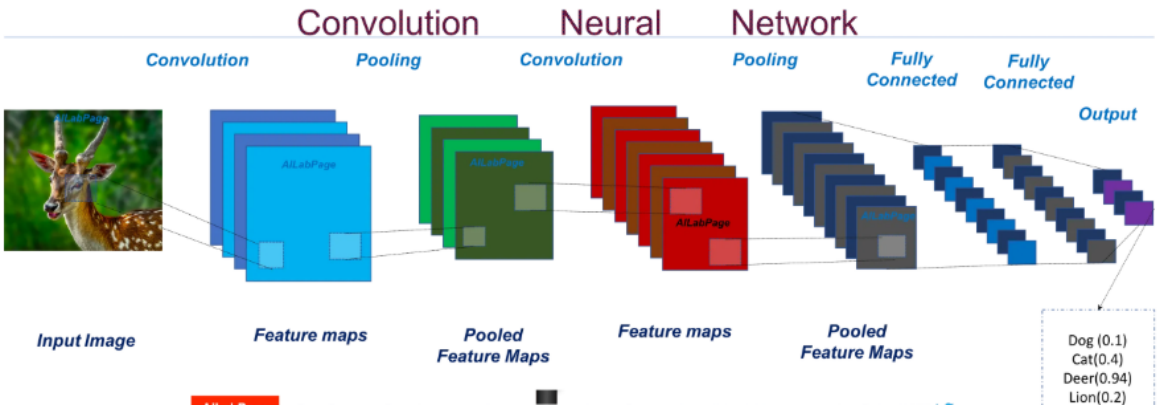


Şekil 3.6: Eğitim Sonuçları.

6 numaralı şekilde gösterildiği üzere yapılan eğitim sonucunda %79,737 oranında başarı elde edilmiştir. Böylelikle ilk eğitim aşaması tamamlanmıştır. Bu aşamadan sonra kıyaslanmak üzere farklı model ve öz nitelikler kullanılarak en iyi model kararlaştırılmıştır.

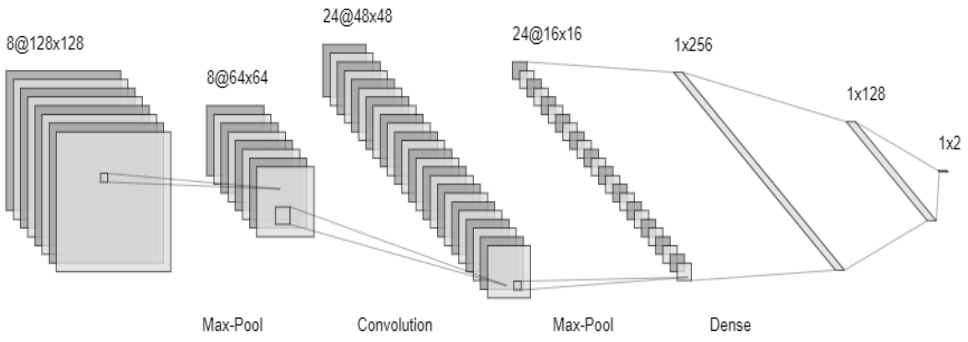
3.3 Evrişimsel Sinir Ağları

Evrişimli sinir ağları, temel olarak görüntüleri sınıflandırmak (örneğin gördüklerini isimlendirmek), benzerlikle kümelemek (fotoğraf arama) ve sahnelerde nesne tanıma yapmak için kullanılan derin yapay sinir ağlarıdır. Evrişimsel sinir ağları yapısı gereği girdi olarak resim ya da videolar alır. Evrişimsel sinir ağları, eğitilebilen birçok katmandan oluşmaktadır. Her katmanın kendi içinde öznitelik havuzlama katmanı, filtre banka katmanı ve doğrusal olmayan katman olmak üzere üç katmanı vardır. Filtre banka katmanında değişik öznitelik çıkarılması işine yarayan birçok çekirdek bulunmaktadır. Havuzlama katmanında, elde edilen her öznitelik haritası ayrı ayrı ele alınır.



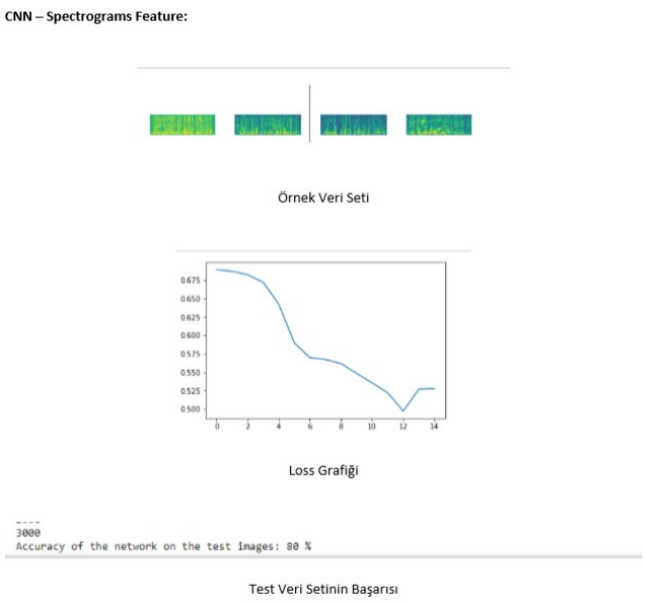
Şekil 3.7: Konvolüsyonel Yapay Sinir Ağı Mimarisi. [12]

Konvolüsyonel Sinir Ağları (Convolution Neural Network-CNN) çok katmanlı bir türüdür. Görme merkezindeki hücreler tüm görseli kapsayacak şekilde alt bölgelere ayrılmış, basit hücreler, kenar benzeri özelliklere, karmaşık hücreler ise daha geniş alıcılarla, tüm görsele yoğunlaştığı düşünülmektedir. İleri yönlü bir sinir ağı olan CNN algoritması da hayvanların görme merkezinden esinlenilerek ortaya atılmıştır. Buradaki matematiksel konvolüsyon işlemi, bir nöronun kendi uyarı alanından uyaranlara verdiği cevap olarak düşünülebilir CNN, bir veya daha fazla konvolüsyonel katman, alt örnekleme katmanı ve bunun ardından standart çok katmanlı bir sinir ağı gibi bir veya daha fazla tamamen bağlı katmandan oluşur [2] Her harita komşu değerinin ortalaması veya maksimum değerinin elde edilmesini sağlamaktadır. Tabi ki resimleri alırken ilgili formata çevrilmiş olması gerekir. Örneğin bir konvolüsyonel sinir ağına bir resim veriyorsak bunu matris formatında vermemiz gerekiyor. Bu aşamada öz nitelik çıkarım bölümünde anlatıldığı üzere ses verisi spectrogram resimlerine dönüştürülerek Evrişimsel sinir ağına matris formatında verilmiştir. Tasarlanan sinir ağı aşağıdaki gibidir.



Şekil 3.8: Evrişimsel Sinir Ağı.

Şekil 3.7’ de verilen model kullanılarak eğitim gerçekleştirilmiştir. Eğitim sonucu elde edilen değerler Şekil 3.8’ de verilmiştir.



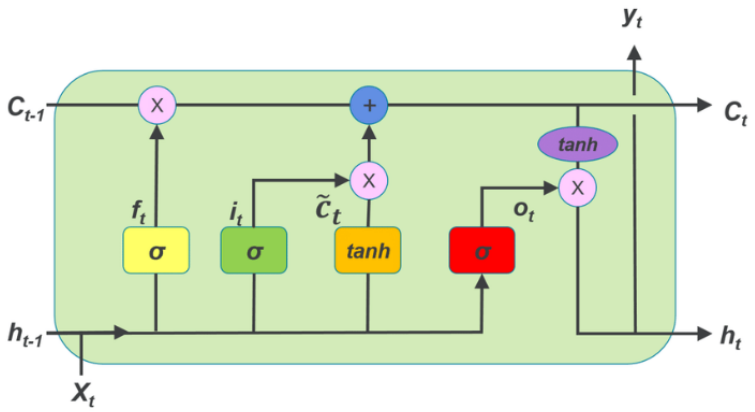
Şekil 3.9: CNN Model Sonuçları.

Şekil 3.8’ de gösterildiği üzere spectrogram öz nitelikleri ile yapılan eğitim sonucu %80 oranında başarı elde edilmiştir. Fakat sinirli sınıfının kesinlik değeri düşük olduğu için i-vector özniteliğine göre daha verimsiz sonuçlanmıştır. Bir sonraki aşamada zaman bazlı öz nitelik olan MFCC öz nitelikleri ile eğitim gerçekleştirilecektir. Bu eğitimin gerçekleşebilmesi için zaman bazlı bir modelin tasarlanması gerekmektedir.

3.4 Uzun-Kısa Süreli Hafıza Modeli (LSTM)

Öz nitelik çıkarım aşamasında anlatıldığı üzerinde MFCC öz niteliği zaman tabanlı öz niteliktir. Bu açıdan eğitim için zaman tabanlı bir model tasarlanmalıdır. Geleneksel sinir ağları ve Evrişimsel sinir ağları böyle bir eğitim gerçekleştirilemez. Örnek vermek gerekirse bir cümleyi anlamlandırırken kelimeleri bağımsız anlamlandırmak yerine her kelime bir önceki kelimeye bağlı olarak anlamlandırır. “Akşam iki adet pizza yedim.” Cümlesinde bulunan kelimeler tek başına ayrı anlamlar taşımaktadır. Fakat kelimeler arasındaki bağlantı oluşturulur ise cümle anlamı oluşmuş olur. Ses verisi de zaman üzerinde değişen veriye

sahiptir. Bundan dolayı tasarlanan modelde bulunan hücreler bu yapıya uygun olmalıdır. Kayıtsız bir şekilde tekrarlayan sinir ağı modelleri RNN olarak adlandırılır. Herhangi bir hafızaya sahip değillerdir. Genellikle konuşma tanıma, dil modelleme, çeviri, resim yazısı gibi alanlarda kullanılır.



Şekil 3.10: LSTM Hücresi. [11]

LSTM ağında ise farklı olarak hafıza yapısı bulunur. LSTM ağları uzun süreli bağımlılıkları öğrenebilen özel bir RNN türüdür. Hochreiter & Schmidhuber (1997) tarafından tanıtıldılar ve aşağıdaki çalışmalarda birçok kişi tarafından rafine edilip popüler hale getirildi. Çok çeşitli problemler üzerinde muazzam bir şekilde çalışıyorlar ve şu anda yaygın olarak kullanılıyorlar. LSTM'ler uzun vadeli bağımlılık sorununu önlemek için açıkça tasarlanmıştır. RNN mimarilerinde önceki bilgi kullanımına dayalı bir yaklaşım vardır. Örneğin "Ağaç toprakta yetişir" cümlesinde "toprak" kelimesini tahmin etmek kolaydır. Fakat bağlamlar arası boşluk arttığında RNN modelin geçmişten gelen bir bilgiyi kullanması oldukça zordur. Örneğin, "İngiltere'de büyüdüm …. Akıcı bir şekilde İngilizce konuşurum." gibi bir metinde "İngilizce" kelimesini tahmin ederken, içinde bulunduğu cümleden yola çıkarak bir dil adı olacağını tahmin edilebilir, ancak doğru kelimenin "İngilizce" olduğunu tahmin etmek için, metnin başındaki cümledeki yer bilgisini hafızada tutmak gerekmektedir. Teoride mümkün olan "uzun-vadeli bağımlılıklar", pratikte büyük problemlere yol açtığı görülmüştür. Bu problemi çözmek için, uzun vadeli bağımlılıkları öğrenebilen özel bir RNN türü olan Uzun Kısa Vadeli Bellek (Long Short Term Memory- LSTM) ağları Hochreiter ve Schmidhuber tarafından 1997 yılında tanıtılmıştır LSTM mimarisi giriş, unutma ve çıkış olmak üzere 3 kapı, blok girişi, Sabit Hata Döngüsü, çıkış aktivasyon fonksiyonu ve gözetleme (peephole) bağlantılarına sahiptir. Bloğun çıktısı tekrar tekrar bloğun girişine ve tüm kapılarına bağlanır. Gözetleme bağlantıları ve unutma kapısı ilk geliştirilen mimaride

bulunmamaktadır. LSTM'in kendi durumunu sıfırlamak için unutma kapısı, kesin zamanlamaları öğrenmeyi kolaylaştırmak için ise gözetleme bağlantıları eklenmiştir. Bilgiyi uzun süreler boyunca hatırlamak pratik olarak onların varsayılan davranışlarıdır, öğrenmeye çalıştıkları bir şey değil. Bu yapı staj süresince MFCC öz niteliği eğitimi için kullanıldı. PyTorch üzerinden tasarlanan model eğitilerek aşağıdaki sonuçlar elde edildi.

```
In [35]: from sklearn.metrics import classification_report
         print(classification_report(pre_label.cpu(), predicted.cpu(

                       precision    recall  f1-score   support

               Normal       0.77      0.69      0.73       119
                Angry       0.61      0.70      0.65        81

            micro avg       0.69      0.69      0.69       200
            macro avg       0.69      0.70      0.69       200
         weighted avg       0.71      0.69      0.70       200
```

Şekil 3.11: LSTM Model Sonuçları.

Şekil 3.8'de verildiği üzere model sonucu ortalama %72 başarı oranına sahiptir. Bunun yanında %70 kesinlik değeri bulunmaktadır.

3.5 Eğitim Sonuçlarının Değerlendirilmesi

İ-vector, spectrogram ve MFCC öz nitelikleri ile uygun modellerde yapılan eğitimlerin sonuçları çizelge 1' de verilmiştir. Bu çizelgede her bir model için ayrı ayrı kayıp ve başarı değerleri yer almaktadır. Bu değerler tek başına değerlendirmek için yetersizdir fakat her modelin başlığı altında kesinlik ve doğruluk değerleri de verilmiştir. Bu değerler değerlendirilirken göze alınan ilk etmen başarı etmenidir. Genel bir durumda başarının %80 civarında olması kabul edilir. Başarı kriterini göz önüne aldığımız vakit LSTM ağı dışındaki diğer modeller yaklaşık olarak %80 başarı ve %50 kayıp değerine sahiptir. Bu durumda bakacağımız bir diğer değer ise kesinlik ve doğruluk değeridir. Bu değerler kıyaslandığı vakit CNN modelinin sinirli sınıfı için düşük (%69) kesinlik değerine sahip olduğu görülmektedir. Bu durumdan ötürü yüksek başarı oranına sahip i-vector modeli kullanılacaktır.

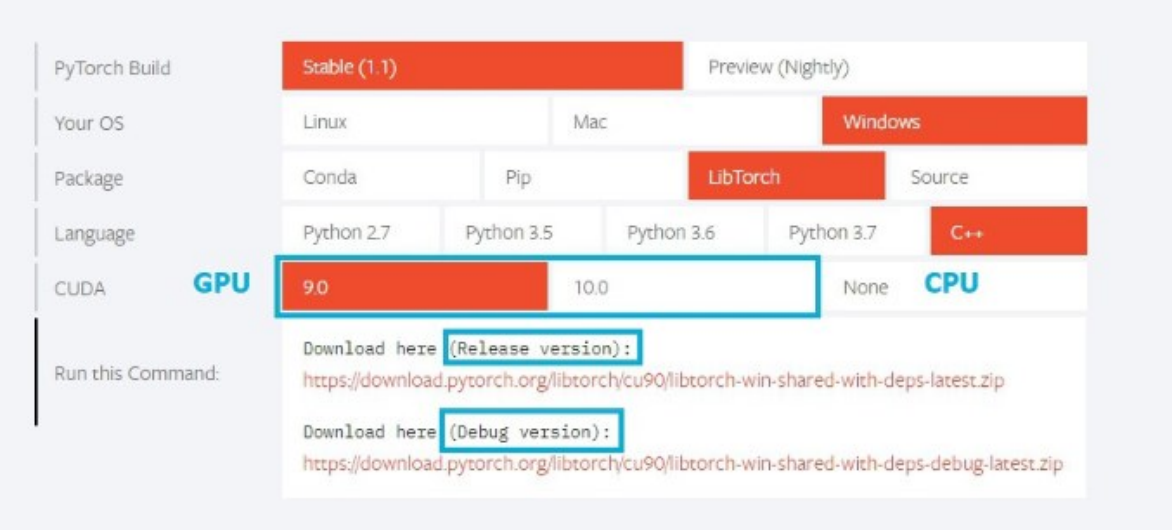
Yöntem	Feature	Loss	Acc	Optimizer
Neural Network 3 Layers	i-Vector	0.475	78.932	SGD / BCEWLOSS
Neural Network 2 Layers	i-Vector	0.531	79.737	SGD / BCEWLOSS
CNN	Spectrograms	0.500	80.000	SGD / CrossEnt.
LSTM	MFCC	0.48	72.000	SGD/ BCELOSS

Train : 10.000 Test: 3000 Veri

Çizelge 3.1: Eğitim Sonuçları

3.6 DLL Dosyasının Yazılması

Bu aşamaya kadar olan adımlarda Python programlama dili ile en optimize model oluşturuldu. Bir sonraki aşamada ise modelin uç kullanıcının kullanabileceği formata getirmektir. Bu doğrultuda modeli farklı ortamlarda çağırabilmemiz gerekmektedir. DLL formatında bulunan dosyalar farklı ortamlarda çağrılabilir. Fakat bunu gerçekleştirebilmek için C++ programlama dilinde modele erişmemiz gerekmektedir. Torch kütüphanesi C++ geliştirme ortamı için libtorch kütüphanesini geliştirmiştir. Stajın geri kalan sürecinde libtorch kütüphanesi kullanılmıştır. Python ortamında eğitilen model katsayıları. pth uzantılı dosyada kayıt edilmiştir. Bu dosya uzantısı torch kütüphanesi tarafından oluşturulmuştur. Dosya modelin optimize katsayılarını içermektedir. Böylelikle model eğitime gerek duymadan tekrar kullanılabilir. Bu aşamadan sonra geliştirme visual studio ortamında gerçekleştirilmiştir. İlk aşamada visual studio ortamına libtorch kütüphanesinin kurulması gerekmektedir. Eğer uygulamamızda CUDA teknolojisi kullanacak isek gpu destekli versiyonunu, kullanmayacak isek CPU versiyonunu indirmemiz gerekmekte.



Şekil 3.12: Libtorch Kütüphanesinin indirilmesi. [9]

Libtorch kütüphanesi bilgisayarda uygun bir klasörde saklanır. Sonraki aşamada ise visual studio üzerinde gerekli ayarlamaların yapılması gerekmektedir. Bu ayarlamalar sırası ile verilmiştir. Bu ayarlar projenin özellikler (properties) kısmında yer almaktadır.

- VC++ dizininde ve C/C++ dizininde bulunan 'Additional Include Directories' yolunana indirmiş olduğumuz klasörde bulunan '\include' ve '\csrc\api\include' yolu verilir.
- Linker dizininde bulunan 'Additional Include Directories' yolunana indirilen klasörde bulunan '\libtorch\lib' yolu verilir.
- Aynı dizinde input yoluna 'torch.lib; caffe2.lib; c10.lib' değerleri girilir.
- Son olarak C/C++ Language sekmesinde bulunana 'Conformance mode' seçeneği 'No' olarak değiştirilir.

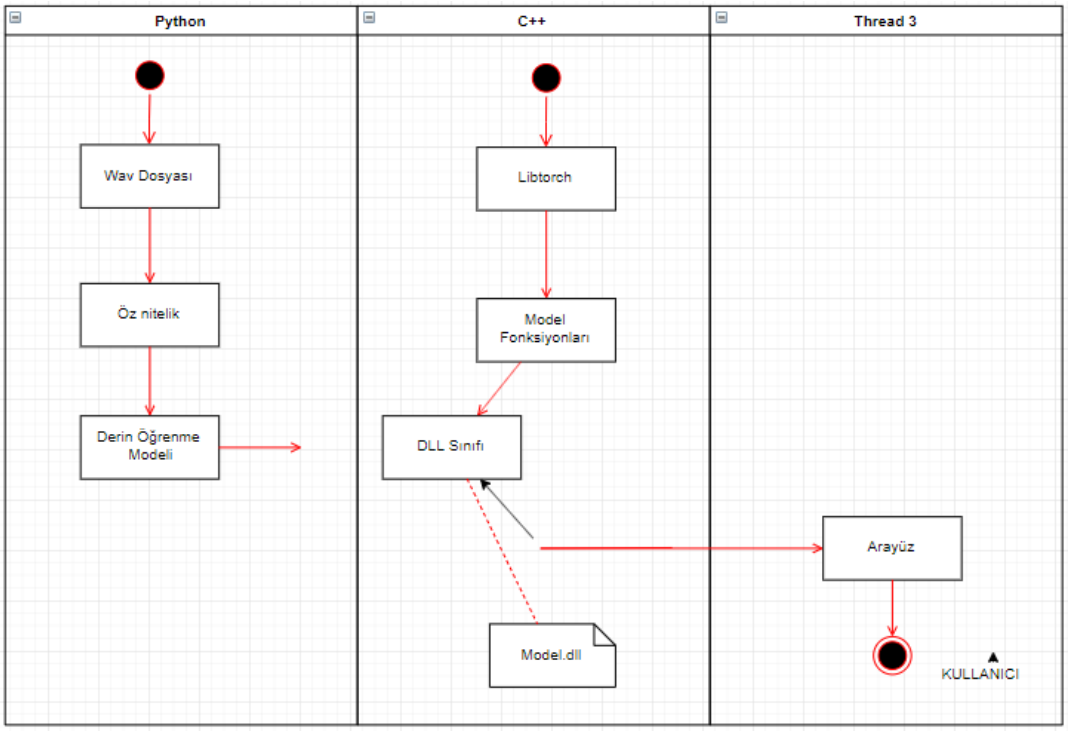
Bu adımlar sırası ile yapıldıktan sonra proje derlenir. Eğer herhangi bir hata alınmaz ise kütüphane başarı ile kurulmuştur. Kütüphane kurulduktan sonra katsayılarını kayıt etmiş olduğumuz model libtorch kütüphanesi üzerinden çağrılır. Böylelikle uygulama artık C++ ortamında kullanılabilir. Bundan sonra uygun fonksiyonlar gerçekleştirilip DLL dosyasının oluşturulması kalır.

```
// MathLibrary.h - Contains declarations of math functions
#pragma once

#ifdef MATHLIBRARY_EXPORTS
#define MATHLIBRARY_API __declspec(dllexport)
#else
#define MATHLIBRARY_API __declspec(dllimport)
#endif

// The Fibonacci recurrence relation describes a sequence F
// where F(n) is { n = 0, a
//               { n = 1, b
//               { n > 1, F(n-2) + F(n-1)
// for some initial integral values a and b.
// If the sequence is initialized F(0) = 1, F(1) = 1,
// then this relation produces the well-known Fibonacci
// sequence: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

// Initialize a Fibonacci relation sequence
// such that F(0) = a, F(1) = b.
// This function must be called before any other function.
extern "C" MATHLIBRARY_API void fibonacci_init(
    const unsigned long long a, const unsigned long long b);
```

Şekil 3.13: Örnek DLL Söz Dizimi.[10]

3.7 Kullanıcı Ara yüzünün Oluşturulması

Son aşamada oluşturulan DLL dosyasının C# ortamında çağrılması yer almaktadır. Bunun için dll dosyasının bulunduğu uzantı uygulamaya verileler 'DLLImport' fonksiyonu yardımı ile dll içinde bulunan fonksiyonlar C# ortamına dahil edilmiş olur. Böylelikle tüm fonksiyonlar C# ortamında kullanılabilir. Uç kullanıcının kullanabilmesi için ara yüz oluşturulur. Oluşturulan bu arayüz ses dosyası seçme ve sınıflandırma butonlarını içermektedir. Ek olarak sonucu gösteren sinir ve normal etiketli iki adet bar yer almakta bu barlar modelin vermiş olduğu sonucu yansıtmaktadır.



Şekil 3.14: Proje UML Diyagramı.

Yukarıdaki şekilde projenin UML diyagramı yer almaktadır. Bu diyagrama göre proje üç katmandan oluşmaktadır. İlk katman Python katman, yapay sinir ağı modelinin oluşturulup eğitildiği katmandır. Bu katmanı sarmalayıcı katman olan C++ katmanı izlemektedir. C++ katmanında libtorch kütüphanesi kullanılmıştır. Bu katmanda projenin DLL dosyası yazılmıştır. Son katman ize kullanıcı tarafında çalışacak olan C# katmanıdır. Bu katmanda ise ara yüz tasarlanarak modelin tüm fonksiyonları ara yüz aracılığı ile çağrılmıştır. Böylece uçtan uca bir ses analiz çözümü sunulmuştur.

4 SONUÇ

Stajımı gerçekleştirmiş olduğum firma ses analizi üzerine uygulamalar geliştirmektedir. Sesli iletişim bulunduğumuz yıllarda hayatımızın hemen her aşamasında yer almaktadır. Sesli iletişim gerçekleştirilirken konuşmacı ile görsel temas kurmak her zaman mümkün değildir. Görsel temas eksikliği jest ve mimiklerin iletilemeyeceği için iletişimi verimsiz kılmaktadır. Proje süresince söz konusu eksiklik dikkate alınarak konuşmacı duygu analizi gerçekleştirilmiştir. Duygu analizinin gerçekleştirilmesi için yapay sinir ağları kullanılmıştır. Proje kapsamında sinirli ve normal olmak üzere iki farklı sınıfta toplamda 12,000 adet ses dosyası toplanmıştır. Bu ses dosyalarından spectrogram, i-vector, MFCC gibi öz nitelikler elde edilmiştir. Daha sonra elde edilen bu öz nitelikler konvolüsyonel sinir ağları, yapay sinir ağları, kısa uzun süreli hafıza modelli sinir ağları ile eğitilmiştir. Yapılan eğitimler başarı oranı, kayıp oranı, doğruluk ve kesinlik değeri parametrelerine göre değerlendirilmiştir. Yapılan bu değerlendirmeler sonucu i-vector ile elde edilen öz niteliklerin yapay sinir ağında eğitilmesi en yüksek başarı oranını vermiştir. Bu doğrultuda staj süresince i-vector modeli kullanılmıştır. Bir sonraki aşamada ise oluşturulan model C++ ortamına aktarılmıştır. Bu aşamada ise libtorch kütüphanesi kullanılmıştır. Böylelikle modelin fonksiyonlarını içeren DLL dosyası oluşturulmuştur. Son olarak ise oluşturulan bu dosya C# programlama dilinde çağrılarak, tüm fonksiyonlara erişim sağlanmıştır. Böylelikle eğitilen model uç kullanıcı tarafından kullanılabilecektir. Son aşamada ara yüz oluşturularak modelin tüm işlemleri uçtan uca ara yüz aracılığı ile kullanılabilir olmuştur.

KAYNAKLAR

[1] J. Salamon and J. Bello, "Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification", *IEEE Signal Processing Letters*, vol. 24, no. 3, pp. 279-283, 2017. Available: 10.1109/lsp.2017.2657381.

[2] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, "The Kaldi Speech Recognition Toolkit," in The Kaldi Speech Recognition Toolkit, 2011

[3]"Planet Of Tunes- How do audio analogue to digital converters work?", *Planetoftunes.com*, 2019. [Online]. Available: http://www.planetoftunes.com/digital-audio/how-do-analogue-to-digital-converters-work.html#.XbNLwegzaiM. [Accessed: 25- Oct- 2019].

[4] I. Ozer, Z. Ozer and O. Findik, "Noise robust sound event classification with convolutional neural network", Neurocomputing, vol. 272, pp. 505-512, 2018. Available: 10.1016/j.neucom.2017.07.021.

[5] "HTK MFCC MATLAB- File Exchange- MATLAB Central", *Mathworks.com*, 2019. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/32849-htk-mfcc-matlab. [Accessed: 25- Oct- 2019].

[6] J. Guo et al., "Deep neural network based i-vector mapping for speaker verification using short utterances", *Speech Communication*, vol. 105, pp. 92-102, 2018. Available: 10.1016/j.specom.2018.10.004.

[7] "Yapay Zekâ Ders Notları 03 | Biyolojik Sinir Sistemi ve Yapay Sinir Ağı Hücresi", *Medium*, 2019. [Online]. Available: https://medium.com/@yasinguzel/yapay-zeka-ders-notlar%C4%B1-03-biyolojik-sinir-sistemi-ve-yapay-sinir-a%C4%9F%C4%B1-h%C3%BCcresi-6555add68d80. [Accessed: 26- Oct- 2019].

[8] "What is a perceptron?", *Medium*, 2019. [Online]. Available: https://towardsdatascience.com/what-is-a-perceptron-210a50190c3b. [Accessed: 26- Oct- 2019].

[9] "PyTorch", *Pytorch.org*, 2019. [Online]. Available: https://pytorch.org/. [Accessed: 27- Oct- 2019].

[10] "İzlenecek yol: Kendi dinamik bağlantı kitaplığınızı oluşturma ve kullanma (C++)", *Docs.microsoft.com*, 2019. [Online]. Available: https://docs.microsoft.com/tr-tr/cpp/build/walkthrough-creating-and-using-a-dynamic-link-library-cpp?view=vs-2019. [Accessed: 27- Oct- 2019].

[11] A. Ismail, T. Wood and H. Bravo, "Improving Long-Horizon Forecasts with Expectation-BiasedLSTM Networks", 2018. [Accessed 27 October 2019].

[12] V. Sharma, "Deep Learning – Introduction to Convolutional Neural Networks | Vinod Sharma's Blog", *Vinod Sharma's Blog*, 2019. [Online]. Available:

https://vinodsblog.com/2018/10/15/everything-you-need-to-know-about-convolutional-neural-networks/. [Accessed: 27- Oct- 2019].

EKLER

EK-1. Staj Başarı Belgesi Bölüm Kopyası

EK-2 Kodlar

```
#!/usr/bin/env python
# coding: utf-8
import os
os.chdir('C:/Users/can.korkut/Desktop/Staj_Dosyalari/emotion-ivectors-3')

normal_vects_train=[]
normal_vects_test = []
angry_vects_train = []
angry_vects_test = []


normal_vec_list = os.listdir('./train/normal/')
for i in normal_vec_list:
   with open('./train/normal/'+i,'r') as f:
      line = f.readline()
      line = line.split()
      line.pop(0)
      line.insert(0,0)
      vects = []
      for i in line:
         float_vec = float(i)
         vects.append(float_vec)
      normal_vects_train.append((vects,[0.999]))


normal_vec_list = os.listdir('./train/angry/')
for i in normal_vec_list:
   with open('./train/angry/'+i,'r') as f:
      line = f.readline()
      line = line.split()
      line.pop(0)
      line.insert(0,0)
      vects = []
      for i in line:
         float_vec = float(i)
         vects.append(float_vec)
      angry_vects_train.append((vects,[0.001]))

normal_vec_list = os.listdir('./test/normal/')
for i in normal_vec_list:
   with open('./test/normal/'+i,'r') as f:
      line = f.readline()
      line = line.split()
      line.pop(0)
      line.insert(0,0)
      vects = []
      for i in line:
         float_vec = float(i)
         vects.append(float_vec)
      normal_vects_test.append((vects,[0.999]))
```

```
normal_vec_list = os.listdir('./test/angry/')
for i in normal_vec_list:
   with open('./test/angry/'+i,'r') as f:
      line = f.readline()
      line = line.split()
      line.pop(0)
      line.insert(0,0)
      vects = []
      for i in line:
         float_vec = float(i)
         vects.append(float_vec)
      angry_vects_test.append((vects,[0.001]))


for _ in range(2):
   angry_vects_train+=angry_vects_train

for _ in range(2):
   angry_vects_test+=angry_vects_test

train_data = angry_vects_train + normal_vects_train


test_data = normal_vects_test + angry_vects_test

import numpy as np

np.random.shuffle(train_data)
#np.random.shuffle(test_data)

train_data,train_label = zip(*train_data)
#test_data,test_label = zip(*test_data)


train_data = np.array(train_data, np.float32)
#test_data = np.array(test_data, np.float32)

from sklearn.model_selection import train_test_split
X_train, x_test, Y_train, y_test = train_test_split(train_data, train_label, test_size = 0.2,
random_state = 0)

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.autograd import Variable

class Net(nn.Module):
   def __init__(self):
      super(Net, self).__init__()
      self.fc1 = nn.Linear(600, 1)
      self.rl1 = nn.Sigmoid()
```

```
    def forward(self, x):
        x = self.fc1(x)
        x = self.rl1(x)
        return x

model = Net()

#device = 'cuda' if torch.cuda.is_available() else 'cpu'
device = 'cpu'
torch.cuda.get_device_name(0)

model.to(device)


criterion = nn.BCEWithLogitsLoss()

optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.1)
#optimizer = optim.Adam(model.parameters(), lr=0.01, betas=(0.9, 0.999))


epoch = 8
losses = []
for epoch in range(epoch):
    running_loss = 0.0
    for i, data in enumerate(X_train, 0):
        inputs = data
        labels = Y_train[i]
        inputs = Variable(torch.FloatTensor(inputs))
        labels = Variable(torch.FloatTensor(labels))
        inputs = inputs.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.data
        if i % 3000 == 0 and i is not 0:
            print ("epoch: {} loss: {}".format(epoch, running_loss/3000))
            losses.append(running_loss/3000)
            running_loss = 0.0


import matplotlib.pyplot as plt
plt.plot(losses)
plt.show()


test_id = 25
print (model(Variable(torch.FloatTensor(x_test[test_id]).to(device))))
print(y_test[test_id])
```

```
pred = model(torch.FloatTensor(x_test).to(device))
pred = (pred>0.5).float()
test_label_tensor = torch.FloatTensor(y_test)
test_label_tensor = (test_label_tensor>0.5).float()
correct = (pred == test_label_tensor.to(device)).float().sum()


from sklearn.metrics import classification_report
print ("Accuracy: {:.3f} ".format(float((correct/test_label_tensor.shape[0]) * 100)))
print(classification_report(test_label_tensor, pred.cpu(), target_names=('Normal','Angry')))

pred = model(torch.FloatTensor(test_data).to(device))
pred_ = (pred>0.5).float()
test_label_tensor = torch.FloatTensor(test_label)
test_label_tensor = (test_label_tensor>0.5).float()
correct = (pred_ == test_label_tensor.to(device)).float().sum()

from sklearn.metrics import classification_report
print ("Accuracy: {:.3f} ".format(float((correct/test_label_tensor.shape[0]) * 100)))
print(classification_report(test_label_tensor, pred_.cpu(), target_names=('Angry','Normal')))

cnt = 0
for i in range(10,90):
   pred = model(torch.FloatTensor(x_test[i]).to(device))
   print(i,pred.data[0] < 0.5,pred.data[0])
   cnt +=1


torch.save(model,"model.pt")


model.to("cpu")

example = torch.rand(600)
traced_script_module = torch.jit.trace(model, example.to("cpu"))

traced_script_module.save('model_jit3.pt')
```