KOCAELİ ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ

NÖRON AĞLARINA GİRİŞ DERSİ PROJESİ

PROJE BAŞLIĞI: YAPAY SİNİR AĞLARI İLE DUYGU ANALİZİ

PROJE YÜRÜTÜCÜLERİ:

110201119 Muhammet GÜR

130201027 Berkay OPAK

130201053 Enes YILDIRIM

130201071 B. Görkem KIZILOK

EĞİTİM KURUMU: Kocaeli Üniversitesi

DANIŞMAN: Prof. Dr. Yaşar BECERİKLİ

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

YAPAY SİNİR AĞLARI İLE DUYGU ANALİZİ

ÖZET

Robot bilgisinin insanlaştırılmasında önemli bir adım, insan operatörünün duygularını sınıflandırma yeteneğidir. Bu yazıda, fiili ifadelerle duygu tanıma kabiliyetine sahip olan zekice bir akıllı sistem tasarımını sunuyoruz. Umut verici sinir ağı iş mimarisi özelleştirilir, eğitilir ve çeşitli sınıflandırma görevlerine tabi tutulur ve bundan sonra performansa göre ağ daha da optimize edilir. Final modelin uygulanabilirliği, kullanıcının duygularını anında geri götürebilecek bir canlı video uygulaması içinde gösterilir.

Anahtar Kelimeler: sinir ağı, duygu analizi, insanlaştırılma,

GİRİŞ

1. YAPAY SİNİR AĞLARININ TANIMI

Yapay sinir ağları canlılarda bulunan sinir sisteminin çalışmasını elektronik ortama taşımayı hedefleyen bir programlama yaklaşımıdır. Yapay sinir ağlarının da canlılarda olduğu gibi öğrenme, hatırlama ve öğrendiklerini güncelleme gibi yeteneklerinin olması hedeflenmektedir.

Sinir sisteminin davranışlarını kopyalayabilmek için yapısının da kopyalanması gerektiğini düşünen bilim adamları yapay sinir ağlarını modellerken de sinir sisteminin yapısını örnek almışlardır.

Yapay sinir hücrelerinin birbirine bağlanmasıyla oluşan bir yapay sinir ağı öğrenme algoritmalarından herhangi birini kullanarak öğrenme sürecini tamamladığında kullanıma hazır hale gelir. Yapay sinir ağı çalıştığı sürece öğrenme ve bilgilerini güncelleme yeteneğine de sahiptir.[1]

2. YAPAY SİNİR AĞLARI KISA TARİHÇESİ

İlk yapay nöron, 1943 yılında nöropsikiyatrist Warren McCulloch ve bilim adamı Walter Pits tarafından üretilmiştir. Ancak dönemin kısıtlı olanakları nedeniyle, bu alanda çok gelişme sağlanamamıştır. Bundan sonra 1969'da Minsky ve Papert bir kitap yayınlayarak, yapay sinir ağları alanında duyulan etik kaygıları da ortadan kaldırmış ve bu yeni teknolojiye giden yolu açmışlardır. İlk gözle görülür gelişmeler ise 1990'lı yıllara dayanmaktadır.[2]

3. YAPAY SİNİR AĞLARININ GENEL ÖZELLİKLERİ

Yapay sinir ağları genel olarak canlı beyninin yapısını gerçekleştirmeyi hedefler. Aşağıdaki işlemleri gerçekleştirebilir:

- Öğrenme
- İlişkilendirme
- Sınıflandırma
- Genelleme
- Tahmin
- Özellik belirleme
- Optimizasyon

Bu işlemleri yapan sinir ağlarının ortak noktası ise bir müdahale yapılmaksızın, elinde bulunan bilgilere göre sonuç üretebilmesidir.
Yapay sinir ağları öğrenme işlemi sırasında verilen bilgiler ile kendini düzenleyerek daha sonraki girdiler için doğru kararlar verebilme yeteneğine sahiptir.

4.YAPAY SİNİR AĞLARININ SINIFLANDIRILMASI

- **Yapılarına Göre Yapay Sinir Ağları**

Yapay sinir ağları içerdiği nöronların birbirine bağlanış şekline göre ileri ve geri beslemeli olarak ikiye ayrılır.

1. **İleri Beslemeli Ağlar** İleri beslemeli ağlarda nöronlar girişten çıkışa doğru düzenli katmanlar şeklindedir. Bir katmandan sadece kendinden sonraki katmanlara bağ bulunmaktadır. Yapay sinir ağına gelen bilgiler giriş katmanına daha sonra sırasıyla ara katmanlardan ve çıkış katmanından işlenerek geçer ve daha sonra dış dünyaya çıkar.
2. **Geri Beslemeli Yapay Sinir Ağları** Geri beslemeli yapay sinir ağlarında ileri beslemeli olanların aksine bir nöronun çıktısı sadece kendinden sonra gelen nöron katmanına girdi olarak verilmez. Kendinden önceki katmanda veya kendi katmanında bulunan herhangi bir nörona girdi olarak bağlanabilir.
   Bu yapısı ile geri beslemeli yapay sinir ağları doğrusal olmayan dinamik bir davranış göstermektedir. Geri besleme özelliğini kazandıran bağlantıların

bağlanış şekline göre geri aynı yapay sinir ağıyla farklı davranışta ve yapıda geri beslemeli yapay sinir ağları elde edilebilir.

- **Öğrenme Algoritmalarına Göre Yapay Sinir Ağları**

Yapay sinir ağlarının verilen girdilere göre çıktı üretebilmesinin yolu ağın öğrenebilmesidir. Bu öğrenme işleminin de birden fazla yöntemi vardır. Yapay sinir ağları öğrenme algoritmalarına göre danışmanlı, danışmansız ve takviyeli öğrenme olarak üçe ayrılır.

1. **Danışmanlı Öğrenme** Danışmanlı öğrenme sırasında ağa verilen giriş değerleri için çıktı değerleri de verilir. Ağ verilen girdiler için istenen çıkışları oluşturabilmek için kendi ağırlıklarını günceller. Ağın çıktıları ile beklenen çıktılar arasındaki hata hesaplanarak ağın yeni ağırlıkları bu hata payına göre düzenlenir.

   Hata payı hesaplanırken ağın bütün çıktıları ile beklenen çıktıları arasındaki fark hesaplanır ve bu farka göre her nörona düşen hata payı bulunur. Daha sonra her nöron kendine gelen ağırlıkları günceller.
2. **Danışmansız Öğrenme** Danışmasız öğrenmede ağa öğrenme sırasında sadece örnek girdiler verilmektedir. Herhangi bir beklenen çıktı bilgisi verilmez. Girişte verilen bilgilere göre ağ her bir örneği kendi arasında sınıflandıracak şekilde kendi kurallarını oluşturur. Ağ bağlantı ağırlıklarını aynı özellikte olan dokuları ayırabilecek şekilde düzenleyerek öğrenme işlemini tamamlar.
3. **Destekleyici Öğrenme** Bu öğrenme yaklaşımında ağın her iterasyonu sonucunda elde ettiği sonucun iyi veya kötü olup olmadığına dair bir bilgi verilir. Ağ bu bilgilere göre kendini yeniden düzenler. Bu sayede ağ herhangi bir girdi dizisiyle hem öğrenerek hem de sonuç çıkararak işlemeye devam eder.

   Örneğin satranç oynayan bir yapay sinir ağı yaptığı hamlenin iyi veya kötü olduğunu anlık olarak ayırt edememesine rağmen yine de hamleyi yapar. Eğer oyun sonuna geldiğinde program oyunu kazandıysa yaptığı hamlelerin iyi

olduğunu varsayacaktır ve bundan sonraki oyunlarında benzer hamleleri iyi olarak değerlendirerek oynayacaktır.

- **Öğrenme Zamanına Göre Yapay Sinir Ağları**

Yapay sinir ağları öğrenme zamanına göre de statik ve dinamik öğrenme olarak ikiye ayrılır.

1. **Statik Öğrenme** Statik öğrenme kuralıyla çalışan yapay sinir ağları kullanmadan önce eğitilmektedir. Eğitim tamamlandıktan sonra ağı istenilen şekilde kullanılabilinir. Ancak bu kullanım sırasında ağın üzerindeki ağırlıklarda herhangi bir değişiklik olmaz.
2. **Dinamik Öğrenme** Dinamik öğrenme kuralı ise yapay sinir ağlarının çalıştığı süre boyunca öğrenmesini öngörerek tasarlanmıştır. Yapay sinir eğitim aşaması bittikten sonra da daha sonraki kullanımlarında çıkışların onaylanmasına göre ağırlıklarını değiştirerek çalışmaya devam eder.

5.YAPAY SİNİR AĞLARININ ÜSTÜNLÜKLERİ

Yapay sinir ağ modelleri biyolojik sinir ağlarının çalışmasından esinlenerek ortaya çıkarılmıştır. Canlılarda bulunan sinir sisteminin modellenmesi sayesinde yapay sinir ağları biyolojik sinir sisteminin üstünlüklerine sahip olmuştur.

- **Doğrusal Olmama** Yapay sinir ağları özellikle doğrusal olmayan sistemlerde tahmin yapma açısından istatistik hesaplamalarına göre daha kolay ve doğru sonuç vermesinden dolayı sık kullanılan bir yöntem haline gelmiştir. Özellikle işletmecilik ve finans alanlarında olmak üzere tahmin gerektiren birçok alanda kullanılmaktadır.

  Yapay sinir ağlarının temel elemanlarından olan yapay sinir hücrelerinin (nöron) doğrusal sonuçlar vermeyişinden dolayı bu özellik ağa da yansımıştır. Doğrusal olmama özelliğinden dolayı yapay sinir ağları karmaşık problemlerin çözümünde de sıkça kullanılmaktadır

- **Paralellik** Klasik problem çözme algoritmalarının aksine yapay sinir ağları paralel çalışmaya uygun bir yapıya sahiptir. Bu özelliği sayesinde çok daha hızlı problem çözebilme yeteneğine sahip olmuştur.
- **Hata Toleransı** Yapay sinir ağları özellikle doğrusal olmayan sistemlerde tahmin yapma açısından istatistik hesaplamalarına göre daha kolay ve doğru sonuç vermesinden dolayı sık kullanılan bir yöntem haline gelmiştir. Özellikle işletmecilik ve finans alanlarında olmak üzere tahmin gerektiren birçok alanda kullanılmaktadır.

  Yapay sinir ağlarının temel elemanlarından olan yapay sinir hücrelerinin (nöron) doğrusal sonuçlar vermeyişinden dolayı bu özellik ağa da yansımıştır. Doğrusal olmama özelliğinden dolayı yapay sinir ağları karmaşık problemlerin çözümünde de sıkça kullanılmaktadır

Bilgisayar üzerinde çalışan bir elemanın zarar görüp devre dışı kalması o elemanın içinde bulunduğu sistemin çalışmamasına neden olur. Ancak paralel çalışabilme özelliği ve yapay sinir hücrelerinin bağımsız çalışabilme yapısından dolayı yapay sinir ağında herhangi bir eleman zarar gördüğünde ağın geri kalanı sorunsuz bir şekilde çalışmaya devam eder. İlk olarak yanlış sonuçlar verebilse de daha sonra yeni yapısını öğrenerek eski performansında çalışmaya devam edebilir.

- **Öğrenebilirlik** Klasik algoritmaların çoğu verilen formüllerin hesaplanması ile aynı girdiler için daima aynı çıktıları üretirler. Lineer olan bu algoritmaların aksine yapay sinir ağları sayesinde programlar öğrenme yeteneği de kazanmışlardır. Klasik algoritmalarda tam olarak tanımlı bir çözüm yolu olmayan problemler çözülemezken yapay sinir ağları sayesinde problemler çözüm yöntemi hakkında herhangi bir bilgi verilmeksizin çözülebilir. Yapay sinir ağlarının bu tip problemleri çözebilmesi için gereken tek şey örnek girdiler için sonuçların verilmesidir.
- **Genelleme** Yapay sinir ağları üzerinde çalıştığı probleme göre eğitildikten sonra eğitim sırasında karşılaşmadığı durumlar için de yanıt verebilir. Örneğin bir satranç taşının görüntüsünün tanıtılmasından sonra bu taşın görüntüsünü içeren ancak gürültülü bir görüntü verildiğinde bile yapay sinir ağı bu taşı tanıyabilir.

- **Uyarlanabilirlik** Yapay sinir ağı üzerinde çalıştığı probleme gör kendini düzenleyerek ağırlıklarını belirler. Bir problemi çözmek için eğitilen yapay sinir ağı herhangi bir başka problemde de kolaylıkla kullanılabilir. Bunun için gereken tek şey yeni problemin girdi ve çıktılarıyla ağın tekrar eğitilmesidir.
- **Hız** Yapay sinir ağları paralel yapısı nedeniyle hızlı bir şekilde çalışıp problem çözme yeteneğine sahiptir. Aynı özelliğinden dolayı donanım üzerinde de kolaylıkla gerçeklenebilir.
- **Analiz ve Tasarım Kolaylığı** Yapay sinir ağlarının temel yapı taşı olan yapay sinir yapısı bütün yapay sinir ağlarında aynıdır. Bundan dolayı yapay sinir hücresinin tasarımından sonra bu temel eleman ile yapay sinir ağları kolaylıkla oluşturulabilir. Yapay sinir ağlarının temel yapısının da aynı olmasından dolayı bu ağlar her türlü problemin çözümünde kullanılabilinir.

6. YAPAY SİNİR AĞLARININ DEZAVANTAJLARI

- **Eğitim Süreci** Yapay sinir ağları oluşturulduklarında hiçbir bilgi içermediğinden dolayı direk olarak kullanılamazlar. Herhangi bir problem çözümünde kullanılacak olan yapay sinir ağının problemde kullanılmadan önce eğitilmesi şarttır. Bu eğitim süresi problemin çözümünden çok daha uzun zaman alabilir.
- **Başlangıç Koşullarına Bağlı Olması** Yapay sinir ağları başlangıç koşullarından bağımsız olarak çok kolay dahi olsa herhangi bir problemi çözemezler. Karar verme anında sadece daha önce öğrendiği koşullara göre sonuç üretebilir. Eğitim sırasında verilen örnekler ağın sonraki problemleri çözmesinde de etkilidir.[3]

7.YAPAY SİNİR AĞLARININ KULLANILDIĞI ALANLAR

Nöron ağları başlıca; Sınıflandırma, Modelleme ve Tahmin uygulamaları olmak üzere, pek çok alanda kullanılmaktadır.

Uzay: Uçuş simülasyonları, otomatik pilot uygulamaları, komponentlerin hata denetimleri vs.

Otomotiv: Otomatik yol izleme, rehber, garanti aktivite analizi, yol koşullarına göre sürüş analizi vs.

Bankacılık: Kredi uygulamaları geliştirilmesi, müşteri analizi ve kredi müraacat değerlendirilmesi, bütçe yatırım tahminleri vs.

Savunma: Silah yönlendirme, hedef seçme, radar, sensör sonar sistemleri, sinyal işleme, görüntü işleme vs.

Elektronik: Kod sırası öngörüsü, çip bozulma analizi, non-lineer modelleme vs.

Eğlence: Animasyonlar, özel efektler, pazarlama öngörüsü vs.

Finans: Kıymet biçme, pazar performans analizi, bütçe kestirimi, hedef belirleme vs.

Sigortacılık: ürün optimizasyonu, uygulama politikası geliştirme vs.

Üretim: üretim işlem kontrolü, ürün dizaynı, makina yıpranmalarının tespiti, dayanıklılık analizi, kalite kontrolü, iş çizelgeleri hazırlanması vs.

Sağlık: göğüs kanseri erken teşhis ve tedavisi, EEG, ECG, MR, kalite artırımı, ilaç etkileri analizi, kan analizi sınıflandırma, kalp krizi erken teşhis ve tedavisi

Robotik: yörünge kontrol, forklift robotları, görsel sistemler, uzaktan kumandalı sistemler, optimum rota belirleme vs.

Dil: sözcük tanıma, yazı ve konuşma çevrimi, dil tercüme vs.

Telekomünikasyon: görüntü ve data karşılaştırma, filtreleme, eko ve gürültü sönümlendirilmesi, ses ve görüntü işleme, trafik yoğunluğunun kontrolü ve anahtarlama vs.

Güvenlik: parmak izi tanıma, kredi kartı hileleri saptama, retina tarama, yüz eşleştirm

8. PROBLEM TANIMI VE ÇALIŞMANIN AMACI

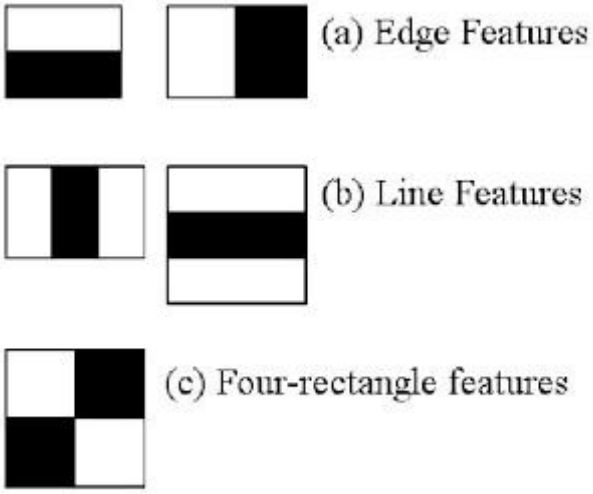
Projede verilen görüntüdeki yüzün tanınması ve bu yüz üzerinden duygu analizinin yapılması gerekmektedir. Bu kısımda yüzün tespitinde bahsedeceğiz.

8.1. Haar Cascades İle Yüz Tespiti

Bu kısımda [4]'üncü kaynak referans alınarak Haar Feature tabanlı Cascade Sınıflandırıcılarını kullanarak yüz algılamanın temellerini göreceğiz.

Haar özellikli çağlayan sınıflandırıcılarını kullanan Nesne Algılama, Paul Viola ve Michael Jones'un kâğıtlarında "Basit Özelliklerin Yaygınlaştırılmış Bir Basamaklandırmasını Kullanan Hızlı Nesne Algılama" adlı çalışmasında 2001'de önerilen etkili bir nesne saptama yöntemidir. Bu, makine öğrenmeye dayalı bir yaklaşımdır; çağlayan fonksiyonu birçok olumlu ve olumsuz görüntülerden eğitim alır. Ardından, diğer görüntülerdeki nesneleri algılamak için kullanılır.

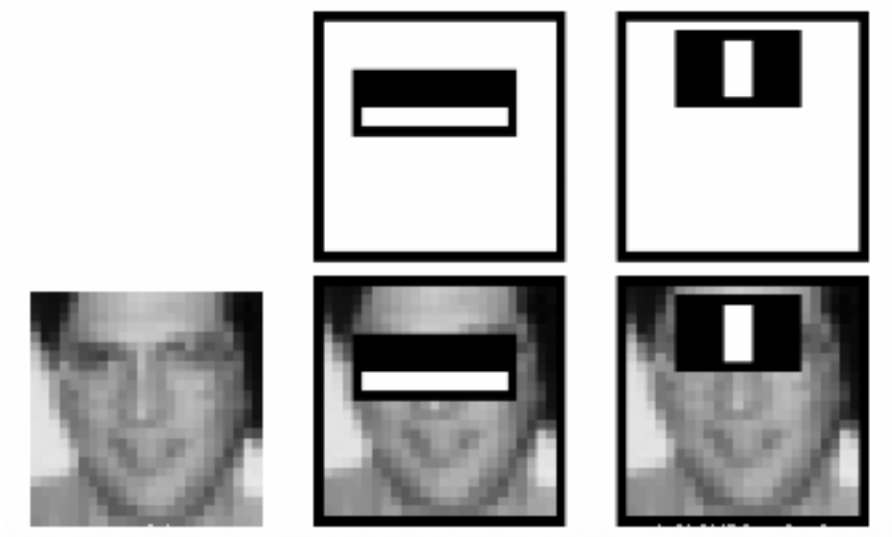
Burada yüz tanımayla çalışacağız. Başlangıçta, algoritmanın sınıflandırıcıyı eğitmek için birçok pozitif resim (yüz resimleri) ve negatif resimler (yüzleri olmayan resimler) gerekir. Ardından, özelliklerini ayıklamalıyız. Bunun için, aşağıdaki resimde gösterilen haar özellikleri kullanılır. Onlar tıpkı konvolüsyon çekirdeğimize benziyorlar. Her özellik, siyah dikdörtgen altındaki piksellerin toplamından beyaz dikdörtgen altındaki piksellerin toplamını çıkararak elde edilen tek bir değerdir.



Şekil 1Haar özelliği

Artık, her çekirdeğin olası tüm boyutları ve konumları, birçok özelliği hesaplamak için kullanılır. (Sadece ne kadar çok hesaplamanın gerekeceğini düşünün, 24x24lük bir pencere bile 160000in üzerinde özellikle sonuçlanır). Her özellik hesabı için, beyaz ve siyah dikdörtgenlerin altındaki piksellerin toplamını bulmamız gerekir. Bunu çözmek için, integral görüntüleri tanıttılar. Piksel toplamının hesaplanmasını basitleştirir, Sadece dört piksel içeren bir işlem için piksel sayısı ne kadar büyük olabilir ki? Güzel, değil mi? Bu işleri çok hızlandırır.

Ancak hesapladığımız bu özelliklerin çoğu alakasızdır. Örneğin, aşağıdaki resmi düşünün. Üstteki satır iki iyi özelliği gösterir. Seçilen ilk kısım, göz bölgelerinin genellikle burun ve yanak bölgelerinden daha koyu olduğunu gösteriyor. Seçilen ikinci kısım, gözlerin burun köprüsünden daha koyu olduğu gösteriyor. Ancak yanaklarda veya başka bir yerde uygulanan çerçevenin alakasız olduğu söylenebilir. Peki, 160000+ özelliğin en iyi özelliklerini nasıl seçebiliriz? İşte bu Adaboost tarafından sağlanır.



Şekil 2Adaboost

Bunun için her eğitim görüntüsüne her bir özelliği uyguluyoruz. Her özellik için yüzleri pozitif ve negatif olarak sınıflandıran en iyi eşiği bulur. Tabii ki, hatalar veya yanlış sınıflamalar olacaktır. Asgari hata oranına sahip özellikleri seçiyoruz; bu, yüzü ve yüz olmayan resimleri en iyi sınıflandıran özellikler anlamına geliyor. (Süreç bu kadar basit değildir Her görüntünün başında eşit ağırlık verilmektedir Her bir sınıflandırmadan sonra, yanlış sınıflandırılmış görüntülerin ağırlıkları artar ve aynı işlem yapılır Yeni hata oranları hesaplanır Ayrıca yeni ağırlıklar da hesaplanır. Gerekli doğruluk veya hata oranı elde edilene veya gerekli özellik sayısı bulunana kadar işlem devam eder).

Nihai sınıflandırıcı, bu zayıf sınıflandırıcıların ağırlıklı bir toplamıdır. Zayıf olarak adlandırılır çünkü yalnız başına görüntüyü sınıflandıramaz, ancak diğerleriyle birlikte güçlü bir sınıflandırıcı oluşturur. Makale, 200 özellikle bile% 95 doğrulukla algılama sağladığını söylüyor. Son kurulumlarında yaklaşık 6000 özellik vardı. (160000+ özellikten 6.000 özelliğe bir düşüş düşünün, bu büyük bir kazançtır).

Şimdi bir görüntü aldın. Her 24x24lük pencereyi al. Buna 6000 özellik uygula. Yüzü olup olmadığını kontrol et. Bu biraz verimsiz ve zaman alıcı bir şey değil mi? Evet öyle. Yaratacıların bunun için iyi bir çözümü var.

Bir resimde, görüntü bölgesinin çoğu yüz bölgesi değildir. Bu nedenle, bir pencerenin yüz bölgesi olup olmadığını kontrol etmek için basit bir yöntem olması

daha iyi bir fikirdir. Değilse, tek bir atışta ilgili kısmı atın. Tekrar işlemeyin. Bunun yerine bir yüz olabilecek bölgeye odaklanın. Bu şekilde, olası bir yüz bölgesini kontrol etmek için daha fazla zaman bulabiliriz.

Bunun için sınıflandırıcıların basamaklandırılması kavramını getirdiler. Bir pencereye tüm 6000 özelliği uygulamak yerine, özellikleri sınıflandırıcıların farklı aşamalarına gruplayıp ve tek tek uygulamalıyız. (Normalde ilk birkaç aşama çok daha az sayıda özellik içerir). Bir pencere ilk etapta başarısız olursa, pencereyi atın. Üzerinde kalan özellikleri düşünmüyoruz. Geçerse, ikinci aşamayı uygulayın ve süreci devam ettirin. Tüm aşamalardan geçen pencere yüz bölgesidir.

Yazarın dedektörü ilk beş aşamada 1, 10, 25, 25 ve 50 özellikli 38 aşamalı 6000+ özelliğe sahiptir. (Yukarıdaki resimdeki iki özellik aslında Adaboost'tan en iyi iki özellik olarak elde edilmiştir). Yazarlara göre, ortalama olarak, alt pencere başına 6000 + 'dan 10 özellik değerlendirildi.

9. PROJEDE (TEZDE) KULLANILAN YÖNTEM VE METODLAR

9.1. TensorFlow

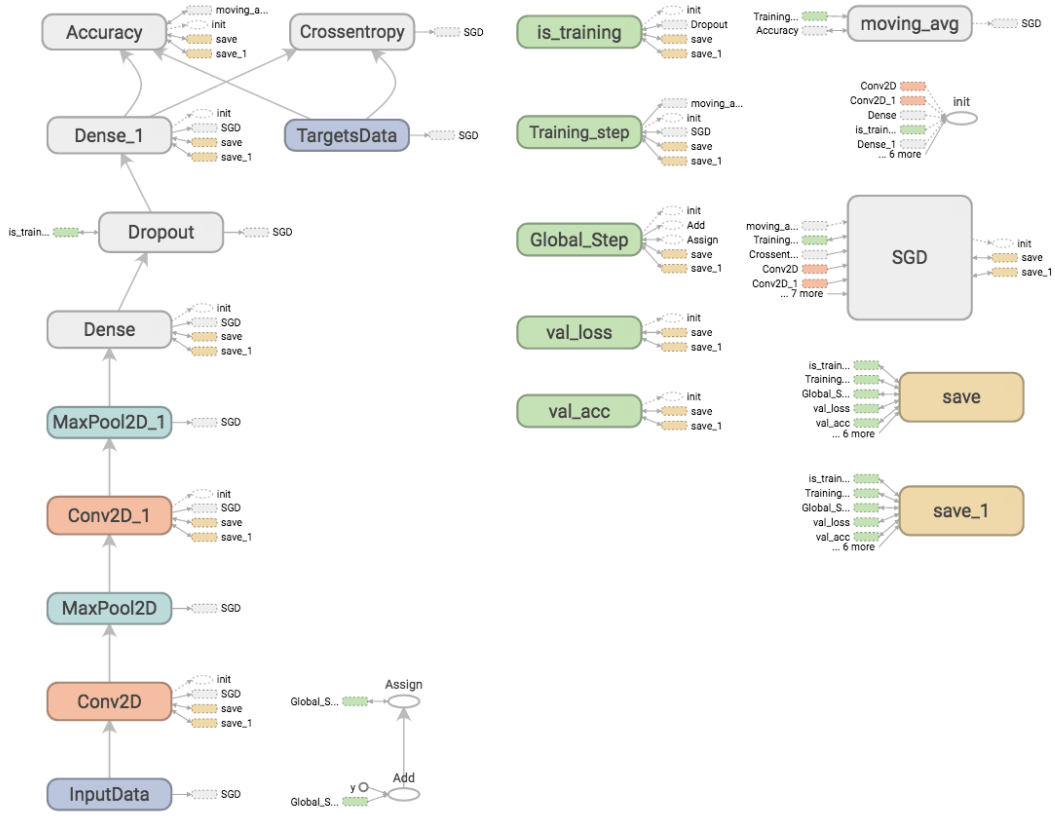
TensorFlow, çeşitli görevlerdeki makine öğrenimi için kullanılan açık kaynak kodlu bir yazılım kütüphanesi olup, insanlar tarafından kullanılan öğrenme ve akıl yürütme yöntemlerine benzer şekilde modeller ve korelâsyonları algılamak ve deşifre etmek için sinir ağları oluşturup eğitim edebilen sistemler için ihtiyaçlarını karşılamak üzere Google tarafından geliştirilmiştir. Şu anda hem araştırma hem de üretim için Google ürünlerinde kullanılmaktadır. TensorFlow başlangıçta Google Brain ekibi tarafından 9 Kasım 2015'te Apache 2.0 açık kaynak lisansı altında serbest bırakılmadan önce dâhili Google kullanımı için geliştirildi.

9.2. TFLearn

TFlearn, Tensorflow'un üzerine kurulmuş, modüler ve şeffaf bir derin öğrenme kütüphanesi. Deneyleri kolaylaştırmak ve hızlandırmak için tam şeffaf ve uyumlu kalırken TensorFlow'a daha üst düzey bir API sağlamak üzere tasarlanmıştır.

TFlearn aşağıdaki özellikleri içerir:

- Son derece modüler dahili sinir ağı katmanları, regülatör, iyileştirici, metrik aracılığıyla hızlı prototipleme ...
- Çoklu giriş, çıkış ve iyileştiricilerin desteğiyle herhangi bir TensorFlow grafiğini eğitmek için güçlü yardımcı fonksiyonlar.
- Öğretici ve örneklerle, derin sinir ağlarının uygulanması için kullanımı kolay ve anlaşılır üst düzey API.



Şekil 3: Model Görselleştirmesi

9.3. Nöron Ağının Oluşturulması

Ağlar, Python'da çalışan TensorFlow'un üstündeki TFLearn kitaplığı kullanılarak programlanır. Bu ortam kodun karmaşıklığını düşürür, çünkü her nöron yerine sadece nöron katmanları oluşturulmalıdır. Program aynı zamanda eğitim ilerlemesi ve aciliyetle ilgili gerçek zamanlı geribildirim sağlar ve eğitimden sonra modeli kaydetmeyi ve yeniden kullanmayı kolaylaştırır. Bu çerçeveyle ilgili daha ayrıntılı bilgi [7] referansında bulunabilir.

```
def build_network(self):
  # Smaller 'AlexNet'
  # https://github.com/tflearn/tflearn/blob/master/examples/images/alexnet.py
  print('[+] Building CNN')
  self.network = input_data(shape = [None, SIZE_FACE, SIZE_FACE, 1])
  self.network = conv_2d(self.network, 64, 5, activation = 'relu')
  #self.network = local_response_normalization(self.network)
  self.network = max_pool_2d(self.network, 3, strides = 2)
  self.network = conv_2d(self.network, 64, 5, activation = 'relu')
  self.network = max_pool_2d(self.network, 3, strides = 2)
  self.network = conv_2d(self.network, 128, 4, activation = 'relu')
  self.network = dropout(self.network, 0.3)
  self.network = fully_connected(self.network, 3072, activation = 'relu')
  self.network = fully_connected(self.network, len(EMOTIONS), activation = 'softmax')
  self.network = regression(self.network,
    optimizer = 'momentum',
```

softmax[i, j] = exp(logits[i, j]) / sum(exp(logits[i]))

Şekil 4Building CNN

9.3.1. Input Data

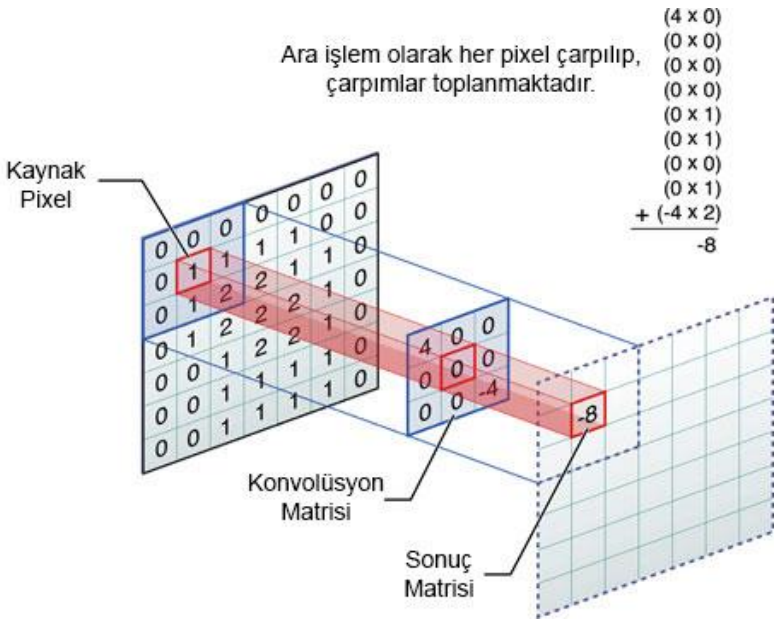
Bu katman, bir ağa veri girmek (ya da beslemek için) için kullanılır. Verilirse, bir TensorFlow yer tutucusu kullanılacaktır, aksi takdirde verilen şekli kullanarak yeni bir yer tutucusu oluşturulacaktır.

Argümanlar

- **shape:** int listesi. Giriş verisinin şeklini temsil eden bir dizi. Hiçbir yer tutucu belirtilmemişse gereklidir. İlk öğe None (toplu boyutu temsil eder) olmalıdır, sağlanmazsa otomatik olarak eklenir.

9.3.2. Conv_2d

Konvolüsyon işlemi görüntü işlemede kare bir resim üzerinde kare bir maskenin sol üst köşeden başlanarak, maskenin merkezi her bir pixel üzerinden geçecek şekilde bütün resmin taranması işlemidir. Bu tarama işlemi sırasında maske içerisinde kalan her bir pixel maskenin katsayıları ile çarpılıp bu çarpımların toplamı, yeni resimde maskenin merkezinin geldiği konuma yazılır. Bu işlem aşağıdaki resimde gösterildiği gibidir. [14]



Şekil 5Konvolüsyon Örneği

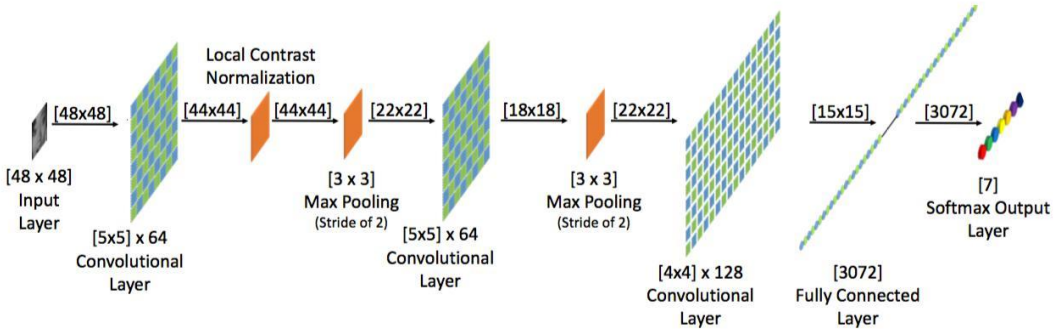
Argümanlar

- **nb_filter:** int. Konvolüsyonel filtre sayısı
- **filter_size:** int. Filtre büyüklüğü
- **activation:** str. Uygulanacak aktivasyon türü. Default ‘linear’

9.3.3. max_pool_2d

Argümanlar

- **kernel_size:** int. Pooling çekirdek büyüklüğü
- **strides:** int. Konv fonksiyonunun stride sayısı



Özgün ağ 48 x 48 girdi katmanıyla başlar ve girilen verinin boyutunu eşleştirir. Bu tabaka sırasıyla bir konvülasyon katmanı, bir yerel zıtlık normalleştirme katmanı ve bir maksimum havuz katmanı tarafından izlenir. Ağ, iki yumuşak katlamalı tabaka ve bir yumuşak-maksimum çıktı katmanına bağlı bir tam kat ile tamamlanır. Bırakma tamamen bağlı tabakaya uygulanmış ve tüm katman ReLu birimleri içermektedir.

Projede parametrelerin sayısını azaltmak için ikinci bir üst üste binme katmanı uygulanmaktadır. Ayrıca öğrenme oranı ayarlanır. Gudi'nin [9] yaptığı gibi öğrenme oranını lineer olarak azaltmak yerine momentumu, eğim aynı yönde ilerlediğinde öğrenme hızını arttırdığından, momentumu kullanan bir öğrenme oranının daha hızlı birleşeceğine inanıyoruz. [11]

Activation=’relu’ kısmında yapay sinir ağında kullanılacak aktivasyon seçiliyor.

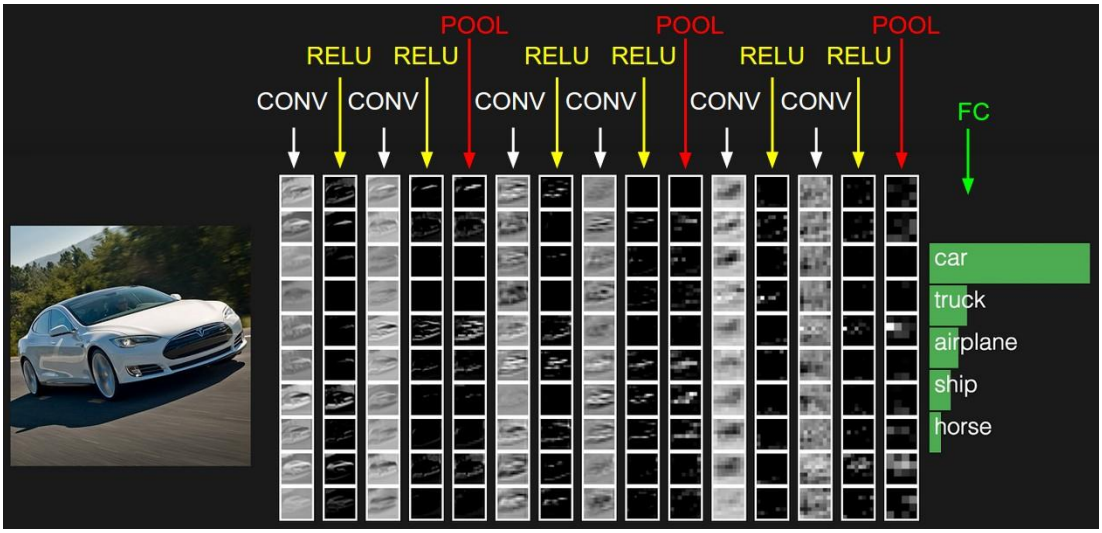
9.3.4. Aktivasyonlar

Genel bir anlamda, ağın sağlamlığını artırmak ve aşırı uyumu kontrol altına almak için doğrusal olmayanlıklar ve boyut korunması sağlarlar.

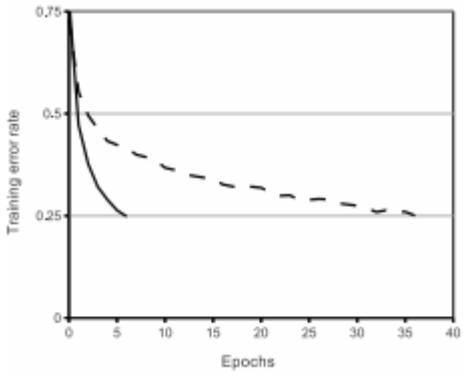
Aktivasyon fonksiyonları, sinir ağlarında kullanılmak üzere farklı doğrusal olmayan tipler sağlar. Bunlar pürüzsüz doğrusal olmayanlar (sigmoid, tanh, elu, softplus ve softsign), sürekli fakat her yerde türemeyen işlevler (relu, relu6, crelu ve relu_x) ve rastgele düzleştirmeyi (bırakma) içerir.[5]

9.3.4.1.ReLU Nonlinearity(Doğrusalsızlık)

Bu katmanın amacı, temel olarak konvektör katmanları sırasında doğrusal işlemler hesaplayan bir sisteme doğrusal olmayanlık getirmektir. ReLU katmanları daha iyi çalışır; çünkü ağ doğruluk açısından önemli bir fark oluşturmadan çok daha hızlı eğitim verebilir. Basit olarak, bu katman tüm negatif etkinleştirmeleri yalnızca 0 olarak değiştirir [13]



Şekil 6ReLU örneği



Şekil 7tnah-ReLU farkı

tflearn.activations.relu (x)

Argümanlar

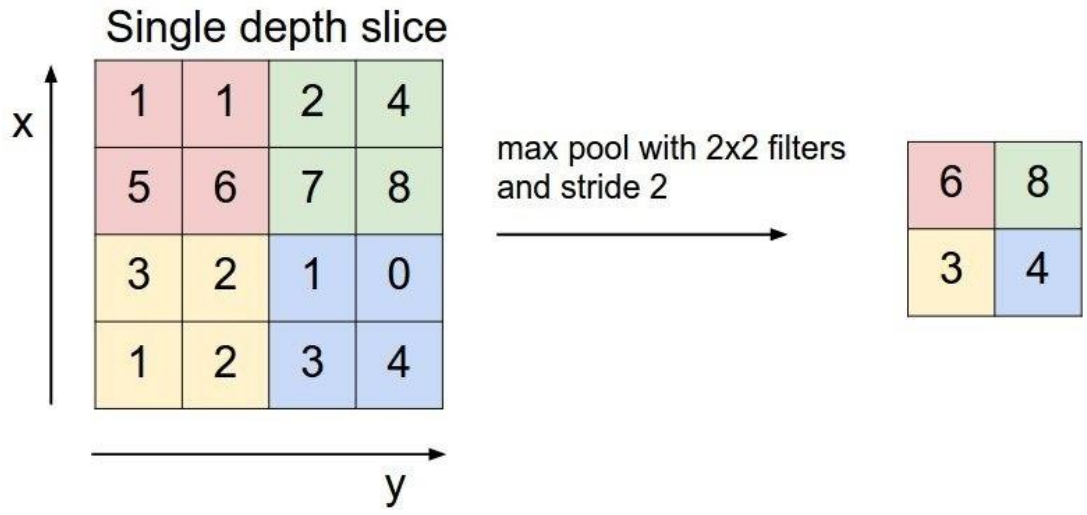
X: Bir Tensör float32, float64, int32, int64, uint8, int16, int8, uint16 değerlerinden olmalıdır.

Returns

Returnunde x değişkeninin aynı tipini döndürür.[6]

9.3.5. Pooling Layers

CNN'lerin bir diğer önemli kavramı, doğrusal olmayan aşağı örneklemenin bir biçimi olan max-pooling'dir. Max-pooling girdi imgesini örtüşmeyen dikdörtgenler kümesine böler ve her bir alt bölge için maksimum değeri verir. Maksimum olmayan değerleri ortadan kaldırarak, üst katmanlar için hesaplamayı azaltır.[15]



Şekil 82x2 Maxpool Örneği

9.3.6. Optimizer

TFLearn tahmincileri ile kullanılmak üzere optimizer yaratmak için temel bir sınıf. İlk olarak, Optimize Edici sınıfı verilen parametrelerle başlatılır, ancak Tensör oluşturulmaz. İkinci aşamada, get_tensor yöntemini çağırmak aslında Tensorflow Optimizer Tensörünü oluşturacak ve geri getirecektir.

Bu şekilde, bir kullanıcı, varsayılan olmayan parametreler ve öğrenme hızı bozulması ile kolayca optimize ediciyi belirleyebilirken, TFLearn tahmincileri optimize ediciyi ve bir basamak tensörünü tek başına inşa edecektir.

Bizim projede kullanıdığımız optimizer metodu '**momentum'**dur.[7]

9.3.6.1.Momentum

```
tflearn.optimizers.Momentum (learning_rate=0.001, momentum=0.9, lr_decay=0.0,
decay_step=100, staircase=False, use_locking=False, name='Momentum')
```

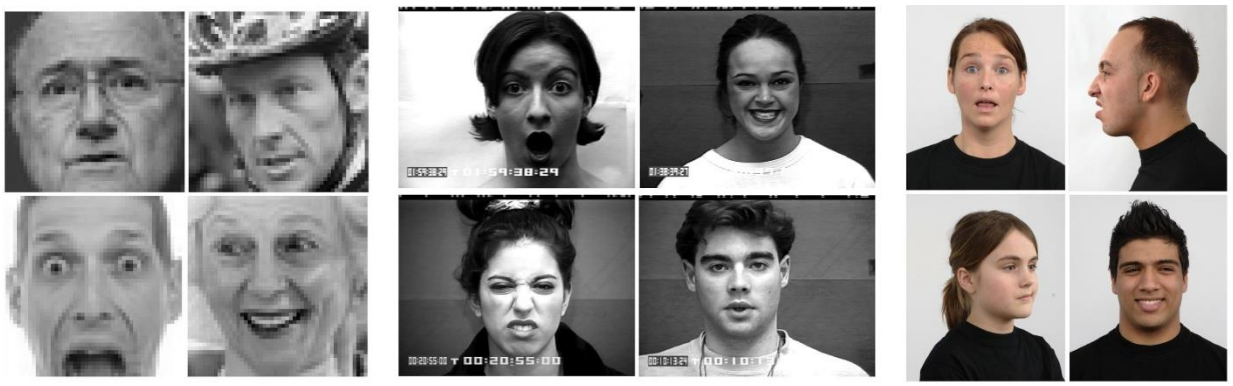
Şekil 9Momentum parametreleri

Momentum optimizerı öğrenme hızı bozunumunu kabul eder. Bir modeli eğitirken, eğitim ilerledikçe öğrenme oranını düşürmek önerilir. İşlev çürük öğrenme oranını döndürür.

```
decayed_learning_rate = learning_rate *  decay_rate ^ (global_step / decay_steps)
```

Şekil 10Çürük öğrenme oranı

9.4. Dataset

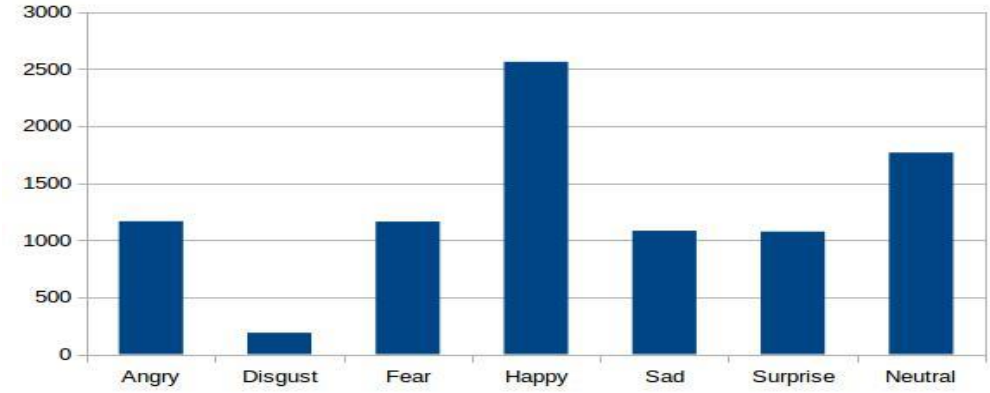


Şekil 11Örnekler FER-2013(sol)-CK+(orta)-RaFD(sağ)

Yapay sinir ağları ve özellikle de derin ağlar, büyük miktarda eğitim verisine ihtiyaç duymaları nedeniyle bilinirler. Üstelik eğitim için kullanılan imajların seçimi, nihai modelin performansının büyük bir bölümünden sorumludur. Bu, hem nitel hem de nicelik bakımından yüksek veri kümesine ihtiyaç duyulduğunu göstermektedir. Duygu tanıma için birkaç yüz yüksek çözünürlüklü fotoğraftan on binlerce küçük görüntüye kadar değişen, araştırma için birkaç veri kümesi mevcuttur.

Projede kullanılan dataset ‘FER-2013 Faces Database’. OpenCV çerçevesinde [10] Haar Özellik Tabanlı Kümelenmiş Klasör kullanılarak tüm veriler önceden

işlenmiştir. Her resim için yalnızca yüzü içeren kare parça alınır, yeniden boyutlandırılır ve 48x48 gri ölçekli bir dizi ile bir dizi haline dönüştürülür.

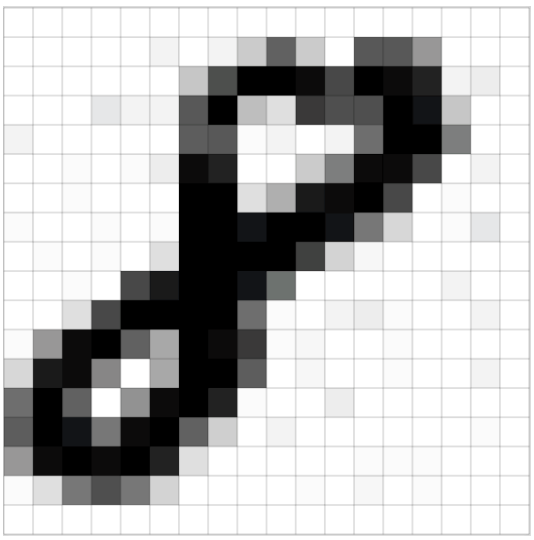


Şekil 12Her duygu eğitimi için mevcut resim sayısı

	emotion	pixels
2	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121 119 115 110 98
3	0	151 150 147 155 148 133 111 140 170 174 182 154 153 164 173 1
4	2	231 212 156 164 174 138 161 173 182 200 106 38 39 74 138 161
5	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 19 43 52 13 26 4
6	6	4 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84 115 127 137 14
7	2	55 55 55 55 55 54 60 68 54 85 151 163 170 179 181 185 188 188
8	4	20 17 19 21 25 38 42 42 46 54 56 62 63 66 82 108 118 130 139
9	3	77 78 79 79 78 75 60 55 47 48 58 73 77 79 57 50 37 44 56 70 8
10	3	85 84 90 121 101 102 133 153 153 169 177 189 195 199 205 207
11	2	255 254 255 254 254 179 122 107 95 124 149 150 169 178 179 17
12	0	30 24 21 23 25 25 49 67 84 103 120 125 130 139 140 139 148 17
13	6	39 75 78 58 58 45 49 48 103 156 81 45 41 38 49 56 60 49 32 31
14	6	219 213 206 202 209 217 216 215 219 218 223 230 227 227 233 2
15	6	148 144 130 129 119 122 129 131 139 153 140 128 139 144 146 1

Şekil 13fer2013.csv içeriği

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0	0
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	107	0	0
0	0	3	0	4	15	236	216	0	0	38	109	247	240	169	0	11	0
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0
6	0	0	4	0	3	252	250	228	255	255	234	112	28	0	2	17	0
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	11	0	0
0	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	9	0
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	1	3	0
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0
6	23	112	157	114	32	0	0	0	0	2	0	8	0	7	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



Şekil 14Pixel Örn1 Şekil 15Pixel Örn2

9.5. Eğitim

TFLearn, eğitim, tahmini, kaydetme / geri yükleme, vb. Gibi otomatik olarak sinir ağı sınıflandırıcısı görevlerini gerçekleştirebilen bir model sarmalayıcı 'DNN' sunar. emotion_recognation.py dosyası içersinde;

9.5.1. Start Training

```
def start_training(self):
  self.load_saved_dataset()
  self.build_network()
  if self.dataset is None:
    self.load_saved_dataset()
  # Training
  print('[+] Training network')
  self.model.fit(
    self.dataset.images, self.dataset.labels,
    validation_set = (self.dataset.images_test, self.dataset._labels_test),
    n_epoch = 100,
    batch_size = 50,
    shuffle = True,
    show_metric = True,
    snapshot_step = 200,
    snapshot_epoch = True,
    run_id = 'emotion_recognition'
  )
```

Şekil 16Eğitim fonksiyonu

Argümanlar

- **X_inputs**: dizi, list dizi listesi (birden fazla girdi olması durumunda) veya dict (giriş katmanı adı anahtarlarla birlikte). Eğitim modeline beslenecek veriler.
- **Y_targets**: dizi, list dizi listesi (birden fazla girdi olması durumunda) veya dict (Anahtar olarak tahmin edicilerin katman adıyla). Eğitim modeline beslenecek veriler.
- **n_epoch**: int. Çalışacak epoch sayısı. Varsayılan: Yoktur.
- **validation_set**: tuple. Doğrulama için kullanılan verileri temsil eder. tuple Veri ve hedefleri tutar (X_inputs ve Y_targets ile aynı türde

sağlanır). Buna ek olarak, eğitim verileri üzerinden veri bölütü gerçekleştirmek için `float` (<1) kabul eder.

- **show_metric**: `bool`. Her adımda gösterim veya tutarlı değil.
- **batch_size**: `int` yada yok. Eğer `int` ise, tüm ağ tahmincilerinin 'batch_size' değerini bu değere göre geçersiz kılar. Ayrıca `validation_batch_size` eğer `int` ise, ve eğer `validation_batch_size` yok ise geçersiz kılınır.
- **shuffle**: `bool` veya yoktur. Eğer `bool` ise, tüm ağ tahmincilerinin 'shuffle' değerini bu değere göre geçersiz kılar.
- **snapshot_epoch**: `bool`. True ise, her dönemin sonunda anlık görüntü modeline geçecektir. (Anlık görüntü bir model doğrulama kümesinde bu modeli değerlendirecek, ayrıca 'checkpoint_path' belirtilmişse bir denetim noktası oluşturacaktır).
- **snapshot_step**: `int` veya None. Eğer `int` ise, her 'snapshot_step' basamağını model olarak anlık görüntüleyecektir.
- **run_id**: `str`. Bu çalıştırma için bir isim verin. (Tensorboard için Faydalı).

9.5.2. Predict

```
def predict(self, image):
  if image is None:
    return None
  image = image.reshape([-1, SIZE_FACE, SIZE_FACE, 1])
  return self.model.predict(image)
```

Şekil 17Tahmin fonksiyonu

Argümanlar

- **X**: dizi, dizi listesi (birden fazla girdi olması durumunda) veya `dict` (katman giriş adlarını tuşlarken). Tahmin için beslenecek veriler.

9.5.3. Load Model

```
def load_model(self):
  if isfile(join(SAVE_DIRECTORY, SAVE_MODEL_FILENAME)):
    self.model.load(join(SAVE_DIRECTORY, SAVE_MODEL_FILENAME))
    print('[+] Model loaded from ' + SAVE_MODEL_FILENAME)
```

Şekil 18Model Yükleme fonksiyonu

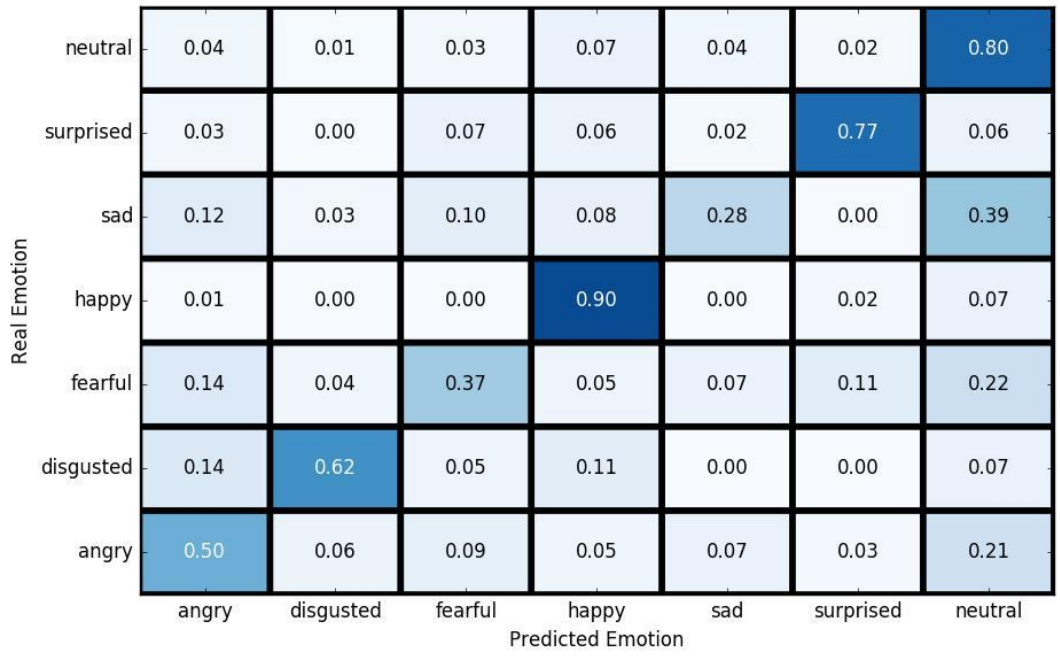
Argümanlar

- **model_file**: str. Model yolu.
- **weights_only**: bool. Doğruysa, yalnızca ağırlıklar geri yüklenir (adım sayısı, hareketli ortalamalar gibi ara değişken değil ...). Toplu normalleştirme kullanıyorsanız ortalamaların da geri yüklenmeyeceğini unutmayın.

python emotion_recognation poc

```
---------------------------------
Run id: MG9PV8
Log directory: /tmp/tflearn_logs/
---------------------------------
Training samples: 1309
Validation samples: 0
--
Training Step: 82  | total loss: 0.64003
| Adam | epoch: 001 | loss: 0.64003 - acc: 0.6620 -- iter: 1309/1309
--
Training Step: 164  | total loss: 0.61915
| Adam | epoch: 002 | loss: 0.61915 - acc: 0.6614 -- iter: 1309/1309
--
Training Step: 246  | total loss: 0.56067
| Adam | epoch: 003 | loss: 0.56067 - acc: 0.7171 -- iter: 1309/1309
--
Training Step: 328  | total loss: 0.51807
| Adam | epoch: 004 | loss: 0.51807 - acc: 0.7799 -- iter: 1309/1309
--
Training Step: 410  | total loss: 0.47475
| Adam | epoch: 005 | loss: 0.47475 - acc: 0.7962 -- iter: 1309/1309
--
Training Step: 492  | total loss: 0.51677
| Adam | epoch: 006 | loss: 0.51677 - acc: 0.7701 -- iter: 1309/1309
--
Training Step: 574  | total loss: 0.48988
| Adam | epoch: 007 | loss: 0.48988 - acc: 0.7891 -- iter: 1309/1309
--
Training Step: 656  | total loss: 0.55073
| Adam | epoch: 008 | loss: 0.55073 - acc: 0.7427 -- iter: 1309/1309
--
Training Step: 738  | total loss: 0.50242
| Adam | epoch: 009 | loss: 0.50242 - acc: 0.7854 -- iter: 1309/1309
--
Training Step: 820  | total loss: 0.41557
| Adam | epoch: 010 | loss: 0.41557 - acc: 0.8110 -- iter: 1309/1309
--
```

Şekil 19Eğitim sırasında sistem çıktısı

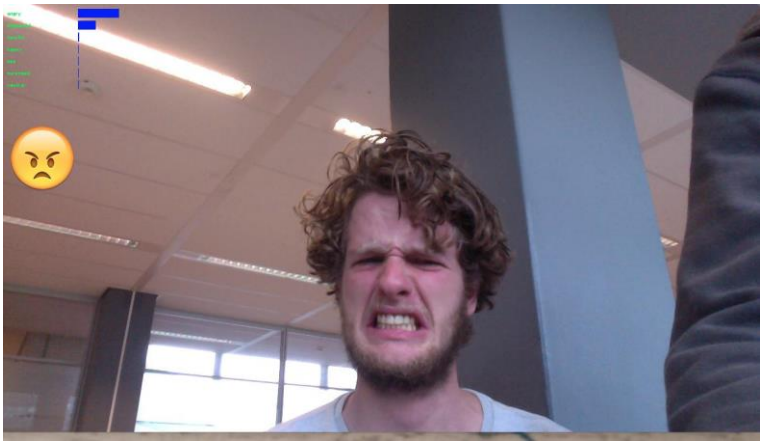


Şekil 20 Modelin performans matrisi.(dikey girdi, yatay çıktı)

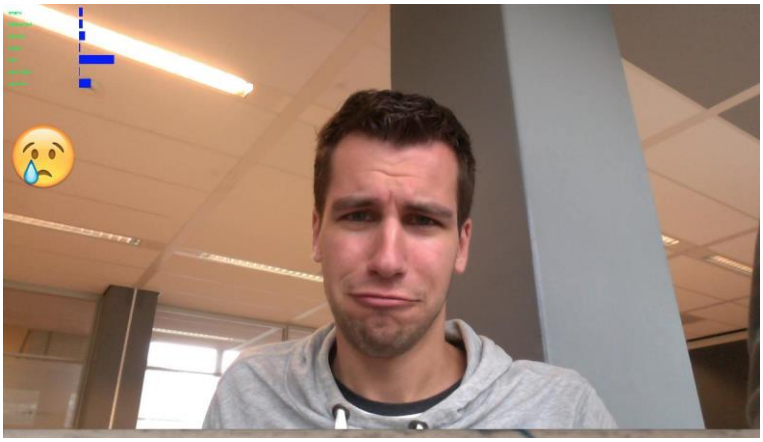
Bu şekildeki ifadede bir duyguyu oluşturan tüm duyguların gerçekteki(dikey) ve tahmin(yatay) edilen oranları görülmektedir. Çok yüksek doğruluk oranları mutlu (% 90), doğal (% 80) ve şaşırmış (% 77) elde edilmiştir. Bunlar aslında insanlara göre en belirgin yüz ifadeleridir. Üzgün, korkulu ve öfkeli olanlar çoğunlukla nötr olarak yanlış sınıflandırılırlar. Görünüşe göre bu duygular birbirine benziyor. En düşük doğruluk üzüntü (% 28) ve korku için (% 37) gözlenir. Son olarak, dikkat çekicidir ki, eğitim setinde tiksindirilmiş etiketli verilerin yüzdesi düşük olsa da sınıflandırma oranı çok makuldür. Genel olarak, doğru sınıflandırmayı gösteren ana diyagonal, açıkça ayırt edilebilir.

10. CANLI UYGULAMA

OpenCV yüz tanıma programı [12] kullanılarak, gerçek zamanlı videodan en büyük görünen yüz izlenmekte, ayıklanmakta ve kullanılabilir 48x48'e ölçeklendirilmektedir. Bu veri daha sonra sinir ağı modelinin girdisine beslenir ve bu da çıkış katının değerlerini verir. Bu değerler, her duygunun kullanıcı tarafından çizilme olasılığını temsil eder. En yüksek değere sahip çıktı kullanıcının duygusu olarak kabul edilir ve ekranın solundaki bir ifadeyle gösterilir.



Şekil 21Canlı uygulama ekran görüntüsü



Şekil 22Canlı uygulama ekran görüntüsü

Şekil 23mp4 video ekran görüntüsü

Şekil 24Canlı uygulama ekran görüntüsü

KAYNAKLAR

[1] http://ahmetkakici.github.io/yazilim/yapay-sinir-aglarina-giris//

[2] *http://docplayer.biz.tr/6797521-Noral-sistemlere-giris-ders-notu.html*

[3] *https://ahmetkakici.github.io/yapay-sinir-aglari/yapay-sinir-aglarinin-siniflandirilmasi//*

[4] http://docs.opencv.org/master/d7/d8b/tutorial_py_face_detection.html.

[5] *https://www.tensorflow.org/versions/r0.11/api_docs/python/nn/activation_functions_*

[6] http://tflearn.org/activations/

[7] *http://tflearn.org/optimizers/*

[8] TFlearn. T earn: Deep learning library fea-turing a higher-level api for tensor ow. URL http://tflearn.org/.

[9] A. Gudi. Recognizing semantic features in faces using deep learning. *arXiv preprint arXiv:1512.00743*, 2015.

[10] OpenSourceComputerVision. Face de-tection using haar cascades. URL http://docs.opencv.org/master/d7/d8b/ tutorial_py_face_detection.html.

[11] What is max pooling in convolutional neural networks? *https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks*

[12] OpenSourceComputerVision. Face de-tection using haar cascades. URL http://docs.opencv.org/master/d7/d8b/ tutorial_py_face_detection.html.

[13] ImageNet Classification with Deep Convolutional Neural Networks *http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf*

[14] *Matlab-goruntu-isleme-konvolusyon-prewitt-ve-sobel-filtreleri* *http://www.erencelik.com/matlab-goruntu-isleme-konvolusyon-prewitt-ve-sobel-filtreleri/*

[15] *DeepLearning* *http://deeplearning.net/tutorial/lenet.html*

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierar-chical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248{255. IEEE, 2009.

[17] T. Ahsan, T. Jabid, and U.-P. Chong. Facial ex-pression recognition using local transitional pat-tern on gaborltered facial images. *IETE Tech-nical Review*, 30(1):47{52, 2013.

[18] B. Fasel and J. Luettin. Automatic facial ex-pression analysis: a survey. *Pattern recognition*, 36(1):259{275, 2003.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classication with deep convolutional neural networks. In *Advances in neural informa-tion processing systems*, pages 1097{1105, 2012.

[20] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classi cation. In *Computer Vision and Pat-tern Recognition (CVPR), 2012 IEEE Confer-ence on*, pages 3642{3649. IEEE, 2012.

[21] Sağıroğlu, Ş., Beşdok, E., Erler, M., 2003. Mühendislikte Yapay Zeka Uygulamaları. İstanbul: Ufuk Yayınevi

[22] R.P.Lippman, "An Introduction to Computing with Neural Nets" IEEE ASP Magazine, 4-22, April 1987

[23] Tek Değişkenli Zaman Serileri Analizine Giriş; Prof. Dr. Süleyman Günay, Dr. Erol Eğrioğlu, Çağdaş Hakan Aladağ; Hacettepe Üniversitesi Yayınları – 2007

[24] Haykin S., Neural Networks, Macmillan Collage Printing Company, New Jersey 1994

[25] Y. Lv, Z. Feng, and C. Xu. Facial expression recognition via deep learning. In *Smart Comput-ing (SMARTCOMP), 2014 International Con-ference on*, pages 303{308. IEEE, 2014.

[26] O. Langner, R. Dotsch, G. Bijlstra, D. H. Wig-boldus, S. T. Hawk, and A. van Knippenberg. Presentation and validation of the radboud faces database. *Cognition and emotion*, 24(8):1377{ 1388, 2010.