



K.K.T.C
YAKIN DOĞU ÜNİVERSİTESİ
Mühendislik Fakültesi

Bilgisayar Mühendisliği Bölümü

LİSANS BİTİRME PROJESİ

Görüntü İşleme İle Captcha Çözümü

20170892 Kutay KELEŞ

Tez Danışmanı
Doç Dr. Melike Şah DİREKOĞLU

Mayıs 2021
Lefkoşa

Etik İlkelerle Uygunluk Beyanı

Bu tezin içinde sunduğum verileri, bilgileri ve belgeleri akademik ve etik kurallar çerçevesinde elde ettiğimi; tüm bilgi, belge, değerlendirme ve sonuçları bilimsel etik ve ahlak kurallarına uygun olarak sunduğumu; çalışmada bana ait olmayan tüm veri, düşünce, sonuç ve bilgilere bilimsel etik kurallar gereği olarak eksiksiz şekilde uygun atıf yaptığımı ve kaynak göstererek belirttiğimi beyan ederim.

Kutay KELEŞ

27/5/2021

Özet

Görüntü İşleme İle Captcha Çözümü

Kutay KELEŞ

Lisans, Mühendislik Bilim Dalı

Tez Danışmanı: Doç Dr. Melike Şah DİREKOĞLU

Projemin genel amacı optik karakter tanıma motoru amacıyla yazılmış, Google tarafından geliştirilen Tesseract-OCR ile günümüzde karşılaştığımız özellikle internet sayfalarında karşılaşılan Captcha resim dosyalarını okutarak resimde bulunan sayı veya harfleri doğru bir şekilde yazıya çevirme amacını taşımaktadır. Kullandığım motorun 3 farklı versiyon ve methodu bulunmakta. İlk iki versiyonu görüntü örtüleme methodu kullanarak resim dosyalarını yazıya çevirmeye çalışır. Fakat ben 2016 yılında sunulan Nöral-Öğrenme methodu ile çalışan versiyonunu kullandım. Yaptığım testler sonucu 20 Dakikalık bir yapay-öğrenme sonucu %90 oranında resimdeki karakterleri doğru şekilde yazıya çevirme işlemi gerçekleşti.

Anahtar Kelimeler: Tesseract,OCR,Optimal Character Recognition,Neural Network,Captcha,Google,Artificial intelligence

Teşekkür

Bu tez çalışmasında emeği geçen hocam Yrd. Doç. Dr. Ümit İLHAN ve proje danışmanım Doç. Dr. Melike Şah DİREKOĞLU'na teşekkür ederim. Ayrıca Google Community'deki gönüllü geliştiricilere yardımlarından dolayı teşekkürlerimi sunarım

Kutay KELEŞ

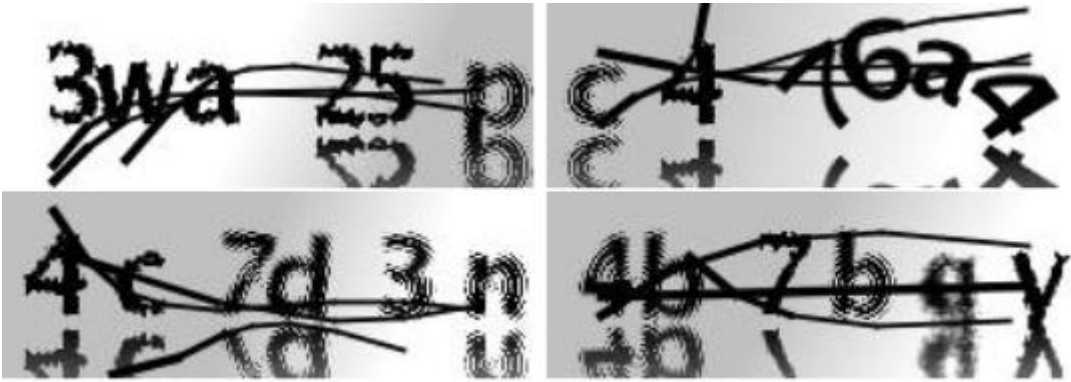
İÇİNDEKİLER

Uygunluk Beyanı	I
Özet	II
Teşekkür	III
İçindekiler	IV
1. Giriş		1
2. Genel Bakış		2
3. Teori	3
3.1. CAPTCHA'lar	5
3.1.1. Yazı Tabanlı Captcha'lar	6
3.1.2. Resim Tabanlı Captcha'lar	7
3.1.3. Video Tabanlı Captcha'lar	8
3.1.4. Ses Tabanlı Captcha'lar	9
3.2. Captcha Değerlendirmesi	10
4. Derin Öğrenme	11
4.1. Long Short-Term Memory(LSTM)	12
4.2. LSTM Mimarisi	13
4.2.1. Single Layer LSTM Ağı	14
4.2.2. Multilayer LSTM Ağı	15
4.2.3. One Dimensional LSTM(1D-LSTM)	18
5. Tesseract-OCR		20
5.1. Tesseract 4.0 Genel Bakış	21
5.1.1. Tesseract ile Entegrasyon	22
5.1.2. Sistem Gereksinimleri	23
5.2. Tesseract Training	24
5.2.1. Gereksinimler	25
5.2.2. Gerçekleştirim	26
5.2.3. Sonuç	27
Kaynaklar	28

1 Giriş

Captcha'lar günümüzde internet sayfalarına yerleştirilen ve genel amacı kullanıcının bir bot yada insan olduğunu anlamak amaçlı kullanılan, insanlar tarafından kolay olarak çözülebilen ama bilgisayarlar tarafından zorlukla çözülebilen bir Turing testi(CAPTCHA) olarak tanımlanabilir.

Captcha'lar güvenlik problemlerinin önüne büyük oranda geçsede yapay zekanın gelişmesiyle yeni yöntemler bulunması şart olacak. Günümüzde Captcha güvenliği olmayan bir kullanıcı tabanlı website düşünürsek ufak bir yazılım parçacığı sayesinde sunucu veritabanına büyük oranda spam yapılabilir. Brute-Force ile kullanıcı şifresi çalınması,DDOS attack gibi büyük güvenlik zafiyetleri ortaya çıkarabilir. Her nasılsa 2FA (İki adımlı doğrulama) ile güvenlik oranı son zamanlarda artırıldı. Karakter tabanlı Captcha'ların doğru cevabı sunucu tarafında tutulduğu için. Görüntü işleme haricinde herhangi bir şekilde makine tarafından çözülemez. Örnek Captcha'lar Şekil1.1 ve Şekil1.2'de görülebilir

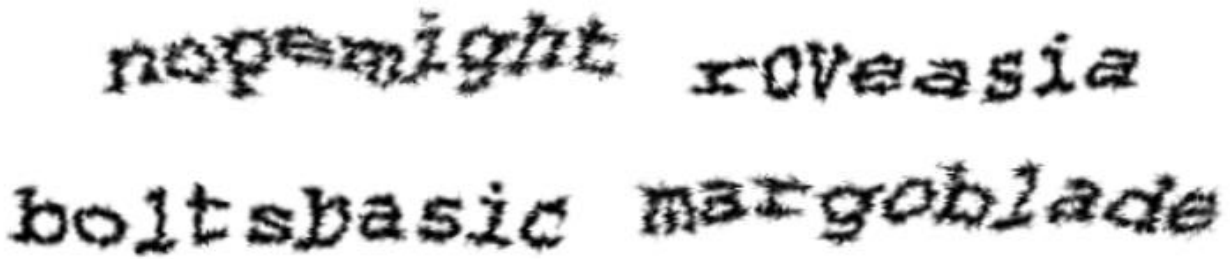


Şekil1.1: www.amazon.com.tr CAPTCHA Örnekleri

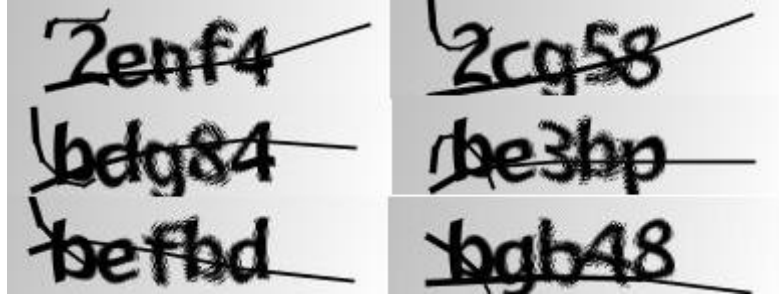
Tesseract-OCR'in temele dayandırdığı (RNN) yapay sinir ağı mimarisini kullanarak LSTM öğrenme çeşidiyle yeni ve daha zor CAPTCHA'lara uyum sağlama yeteneklerini ve öğrenme sonucunda ne aktif öğrenmenin ne kadar yardımcı olacağını keşfetmek istiyoruz.



Şekil1.2: www.gmx.net CAPTCHA örnekleri



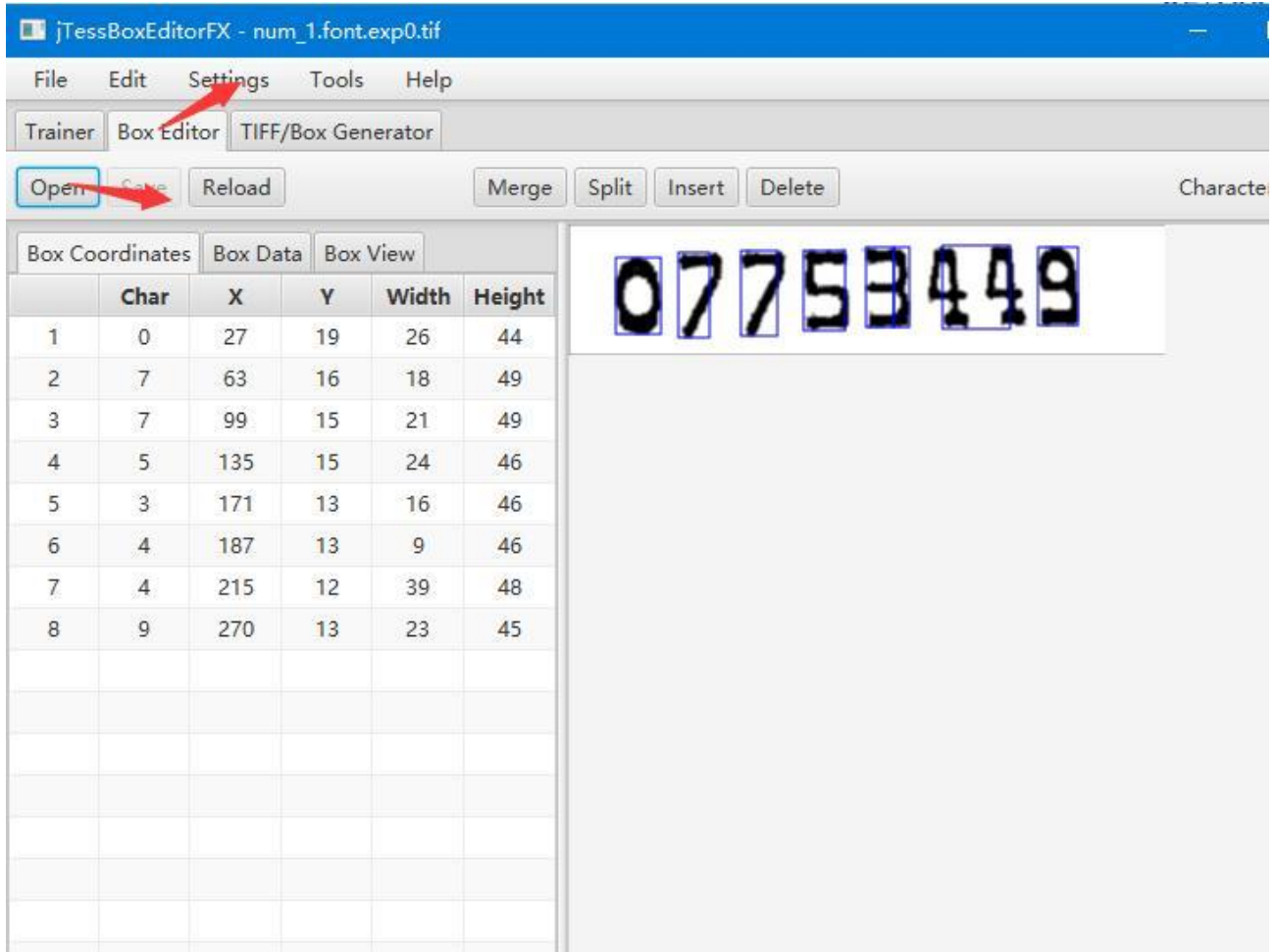
Şekil 1.3: www.wikipedia.org CAPTCHA örnekleri



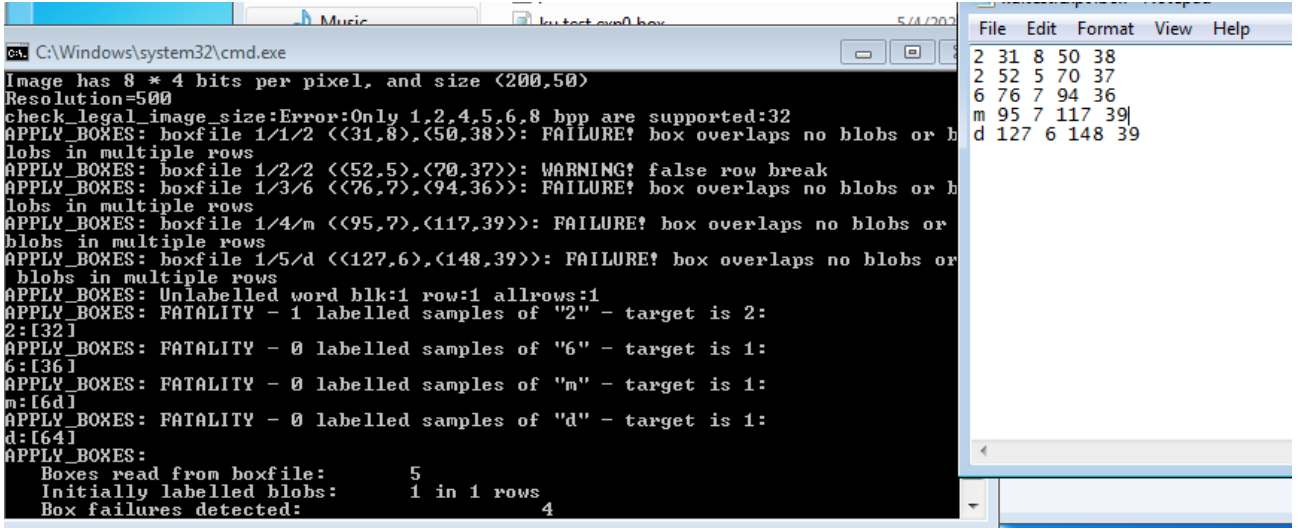
Şekil1.4: Projede işleyeceğimiz CAPTCHA örnekleri

2 Genel Bakış

Karakter tabanlı CAPTCHA'lar yıllar boyunca ücretsiz e-posta sağlayıcılar tarafından kötü amaçlı yazılımlardan korunma amaçlı kullanılmıştır. Doğal olarak programların CAPTCHA'ları çok az veya hiç insan müdahalesi olmadan otomatik olarak çözme girişimleri olmuştur. Derin öğrenme gelişmeden önceki yıllarda CAPTCHA'ları çözmek için görüntü örtüleme kullanılıyordu. Karakterleri ve kelimeleri yerelleştirmek ve onları tanımak için algoritmalar kullandılar. Tesseract-OCR v3 bu şekilde çalışır. Örnek vermek gerekirse elimizde 100 tane örnek Dataset olduğunu varsayalım. Her veri seti için karakterleri belirli bir koordinasyon şeklinde tanıtarak algoritmaya göre öğretilmeye çalışılıyordu. Görüntü örtüleme'ye örnek resim 1.1 de gösterilmiştir.



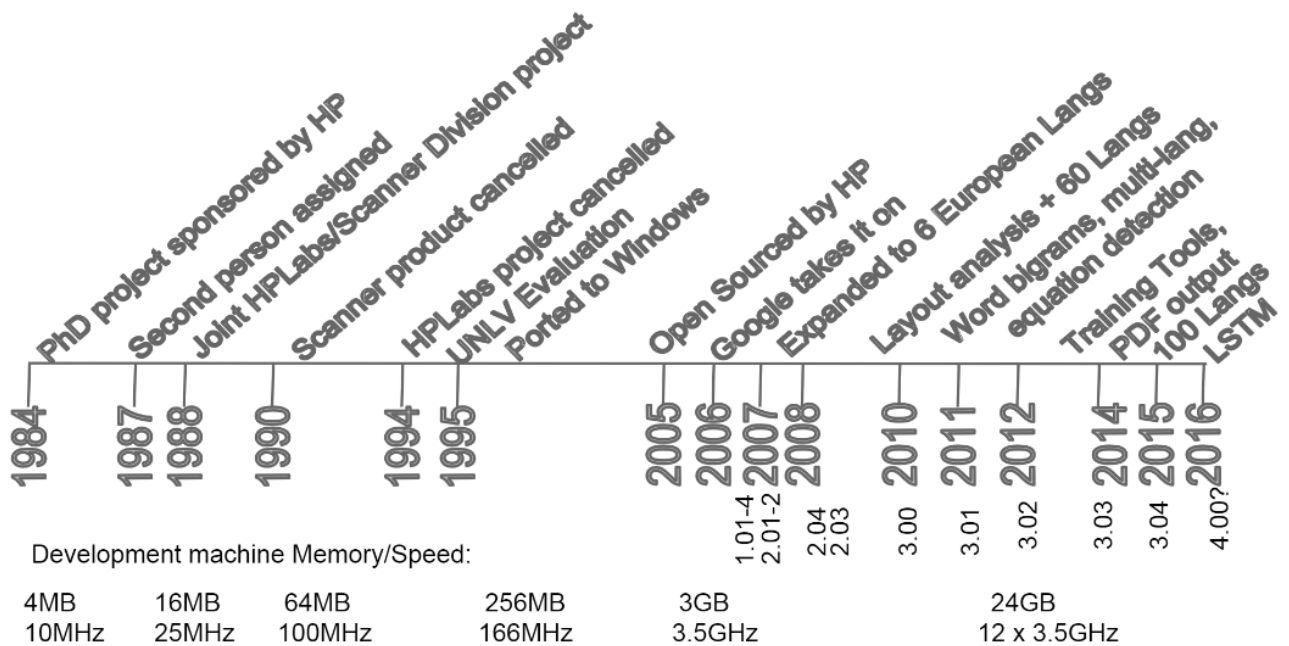
Resim 1.1: jTessBoxEditor kullanılarak karakterlerin koordinasyonları belirtilmiştir.



Resim 1.2: Şekilde karakter,koordinasyonları belirtilmiş bir box'u CNN ile training etme gösterilmiştir.

Gelişen teknolojiyle beraber, Tesseract 4.0 yayınlandı. Yapılan çalışmalarda 3.0'dan daha tutarlı olduğu gözlemlendi. Daha önce söz ettiğimiz gibi Tesseract 4.0 yapay öğrenme modeli kullanıyor. Long Short-Term Memory(LSTM), RNN mimarisi kullanır

Tesseract Timeline



3 Teori

Bu bölümde, projeyi anlamak için gerekli teorik temelleri açıklayacağız. Sinir ağları, CNN'ler ve CAPTCHA'lar uzun süredir bulunduğundan dolayı materyaller açısından bir çok kaynak bulabiliriz. Derin öğrenme teorisini ve LSTM sinir ağı mimarisini 4. Bölümde göreceğiz. Bu bölümde Captcha türleri ve yapılarından bahsedeceğiz.

3.1 Captcha'lar

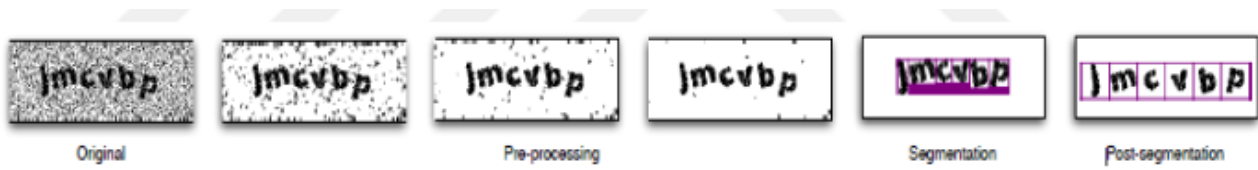
Günümüzde Captcha'lar birçok versiyon halinde sistemlere entegre ediliyor. Her birinin avantaj ve dezavantajı bulunmakta, bazıları kolay entegre edilebilirken kolay kırılabilir.

3.1.1 Yazı Tabanlı Captcha'lar

Teknolojinin gelişmesiyle her ne kadar sistem güvenlikleri artsada aynı şekilde saldırganların bu yöntemleri kırmasında kolaylaşıyor. Bu yüzden geliştiriciler Captcha'lara günden güne komplike ederek saldırganların işini zorlaştırıyor.

Geliştiriciler tarafından yazı tabanlı Captcha'lara farklı pattern'ler uygulayarak bunları çözülmede zorlaştırmaya çalışmakta. Fakat aynı zamanda saldırganlar çeşitli methodlar uygulayarak resimdeki karakterleri daha kolay tanınabilir hale getiriyor. Yazıları algılamayı kolaylaştırmak için saldırganlar resimlere preprocessing uygular.

- Pre-Processing: Bu bölümde görüntü iyileştirme yapılır. Örneğin Python'un opencv kütüphanesini kullanarak captcha daha kolay okunabilir hale getirilebilir.



Resim 3.1 Preprocessing Örneği

Her nasılsa artık yapay zekanın gelişmesiyle birlikte pre-processing'e gerek kalmadan yeterli sayıda dataset ile öğrenme gerçekleştiriliyor. Sonuç olarak günümüz için kullanılması tavsiye edilmeyen bir captcha türü olmuştur.

3.1.2 Resim Tabanlı Captcha'lar

Resim tabanlı Captcha'larda kullanıcılardan verilen örnek kelimeye uygun resimleri seçmesi istenir. Örnek resim3.2 aşağıda verilmiştir.



Resim 3.2: Resim Tabanlı Captcha

Bu captcha türü günümüz için başarılı bir güvenlik yöntemi sağlamaktadır. İnsanlar verilen kelimeyi kolayca algılayabilirken bilgisayarın anlaması için gelişmiş bir yapay öğrenme çalışması gerekir.

3.1.3 Video Tabanlı Captcha'lar

Video tabanlı captcha'lar henüz gelişen teknoloji olduğundan dolayı pek fazla karşılaşılmaz. Kullanıcıdan verilen kısa videoyu izlemesi ve videonun ne ile alakalı olduğundan kısa kelimelerle açıklaması beklenir. Örnek resim aşağıda verilmiştir.



Resim 3.3 Video tabanlı Captcha

Resim3.3 de gösterildiği üzere videoda 2 adet kostümlü köpek bulunuyor. Verilen cevap “dogs costume halloween” gönderildiği zaman sunucu tarafındaki algoritma verilen cevabı inceleyerek doğru sonuç olup olmadığını algılayacaktır.

Video- tabanlı captcha her ne kadar güvenilir dursada sunucu maliyeti yüksek olacağından (memory management) şuanlık talep görülen bir CAPTCHA türü değildir. Ayrıca tabiki farklı dil bilen kullanıcılar içinde büyük bir problem yaratacaktır.

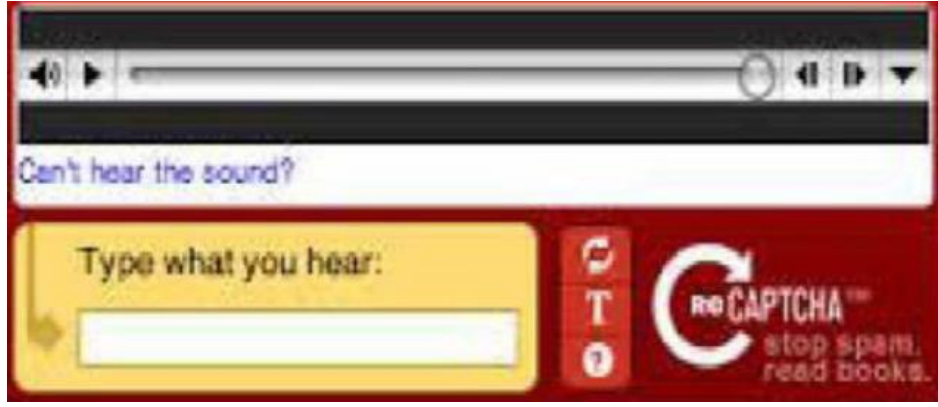
3.1.4 Ses Tabanlı Captcha’lar

Ses tabanlı captcha’lar günümüzde geçerliliğini hemen hemen yitiren türdür. Çünkü sesi yazıya dönüştüren sistemler olduğu gibi kullanıcıların kullandığı dil farklılığı yüzünden büyük bir problem teşkil etmektedir.

Bu captcha türü aşağıdaki yolu takip ederek işler:

- Kullanıcı CAPTCHA sesini dinler
- Kullanıcı duyduğu harfleri baştan sona şekilde yazar.
- Eğer doğru kelimeyi yazarsa işlem gerçekleştirilir

Aşağıdaki resimde örnek verilmiştir.



Resim 3.4: Ses Tabanlı Captcha

3.2 Captcha Türleri Değerlendirmesi

Tablo 3.3. CAPTCHA Değerlendirmesi

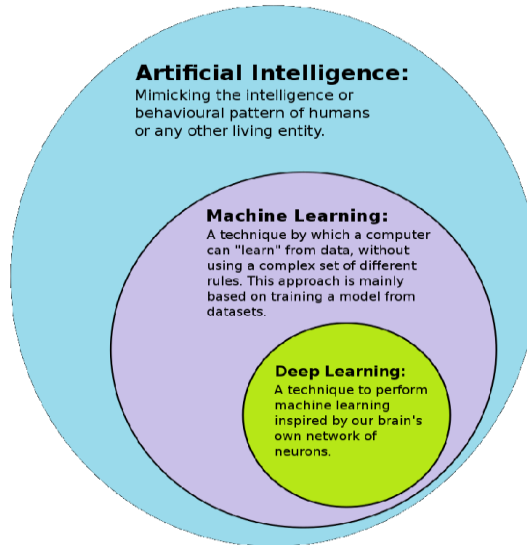
Captcha Türleri	Avantaj	Dezavantaj
Yazı Tabanlı Captcha	<ul style="list-style-type: none"> Kolay Entegre Kullanıcı Dostu 	<ul style="list-style-type: none"> Kompleks testlerde kullanıcı problem yaşayabilir. OCR teknikleri ile kolayca çözülebilir
Resim Tabanlı Captcha	<ul style="list-style-type: none"> Çok zor kırılır. Kullanıcı Dostu 	<ul style="list-style-type: none"> Kullanıcılar algılaması zor resimlerde problem yaşayabilir
Video Tabanlı Captcha	<ul style="list-style-type: none"> Kırılması zordur, gelişmiş teknik gerekir. 	<ul style="list-style-type: none"> Sunucu maliyeti Kullanıcı dostu değil. Rahatsızlık yaratabilir
Ses Tabanlı Captcha	<ul style="list-style-type: none"> Kırılması zordur, Ses tanıma yazılımı gerekir 	<ul style="list-style-type: none"> İngilizce gerektirir. Sese benzer karakterler olabilir

4 Derin Öğrenme(Deep Learning)

Derin öğrenme yapay sinir ağlarına dayalı makine öğrenimi metodları ailesinin bir parçasıdır. Öğrenme gözetimli,yarı gözetimli yahut gözetimsiz gerçekleşebilir.

Derin öğrenme mimarileri çeşitlerinden bazıları olan derin sinir ağları,konvolüsyonel sinir ağları gibi mimariler günümüzde kullanıldığı alanlar; görüntü işleme,ses tanıma,doğal dil işleme,makine dönüşümü,ilaç tasarımı,tıbbi resim analizi,masa oyunları(satranç vb.) olarak tanımlanabilir. Öğrenme sonucunda çoğu alanda insan performansının üstüne çıktığı görülür.

Yapay sinir ağlarının gelişiminde insan beynindeki işlem ve iletişim düğümlerinden ilham alınmıştır. Yapay sinir ağlarının insan beyninden farkı statik ve sembolik olma eğilimindedir. Canlı organizmaların beyni dinamik ve analog şeklinde çalışır.



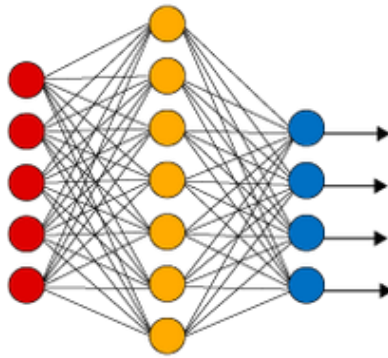
Resim 4.1: Derin öğrenme makine öğrenmenin alt kümesidir.

4.1 Long Short-Term Memory(LSTM)

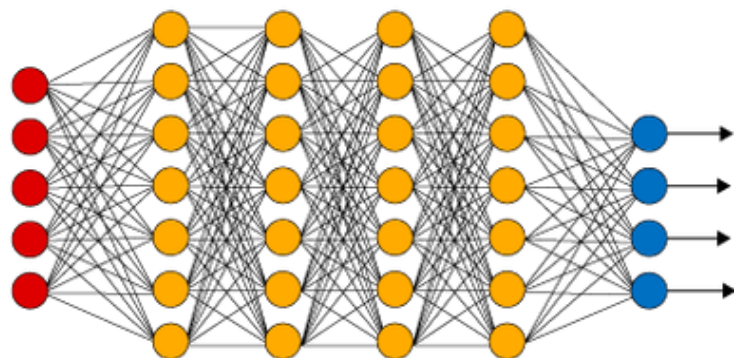
Bu bölümde Long Short-Term Memory(LSTM) ne olduğu ve nasıl çalıştığı hakkında bilgi edineceğiz. İlk olarak LSTM ağlarının teknik olarak nasıl matematiksel olarak formülize edildikleri hakkında konuşacağız. İkinci olarak LSTM ağlarının hangi alanlarda sıkça kullanıldığını göreceğiz. Projenin kapsamı olan Optimal Character Recognition(OCR) dahil. Son kısımda LSTM ağlarını kullanırken hangi parametleri kullanmamız gerektiğinden bahsedeceğiz.

LSTM ağları Recurrent Neural Networks(Yinelenen Sinir Ağları)'un yeni bir mimarisidir. Yinelenen sinir ağları, ileribesleme sinir ağlarından farklı olarak sinirler hidden layer(gizli katman)'ların kendine bağlı olması(self-connected) şeklinde çalışır.

Simple Neural Network



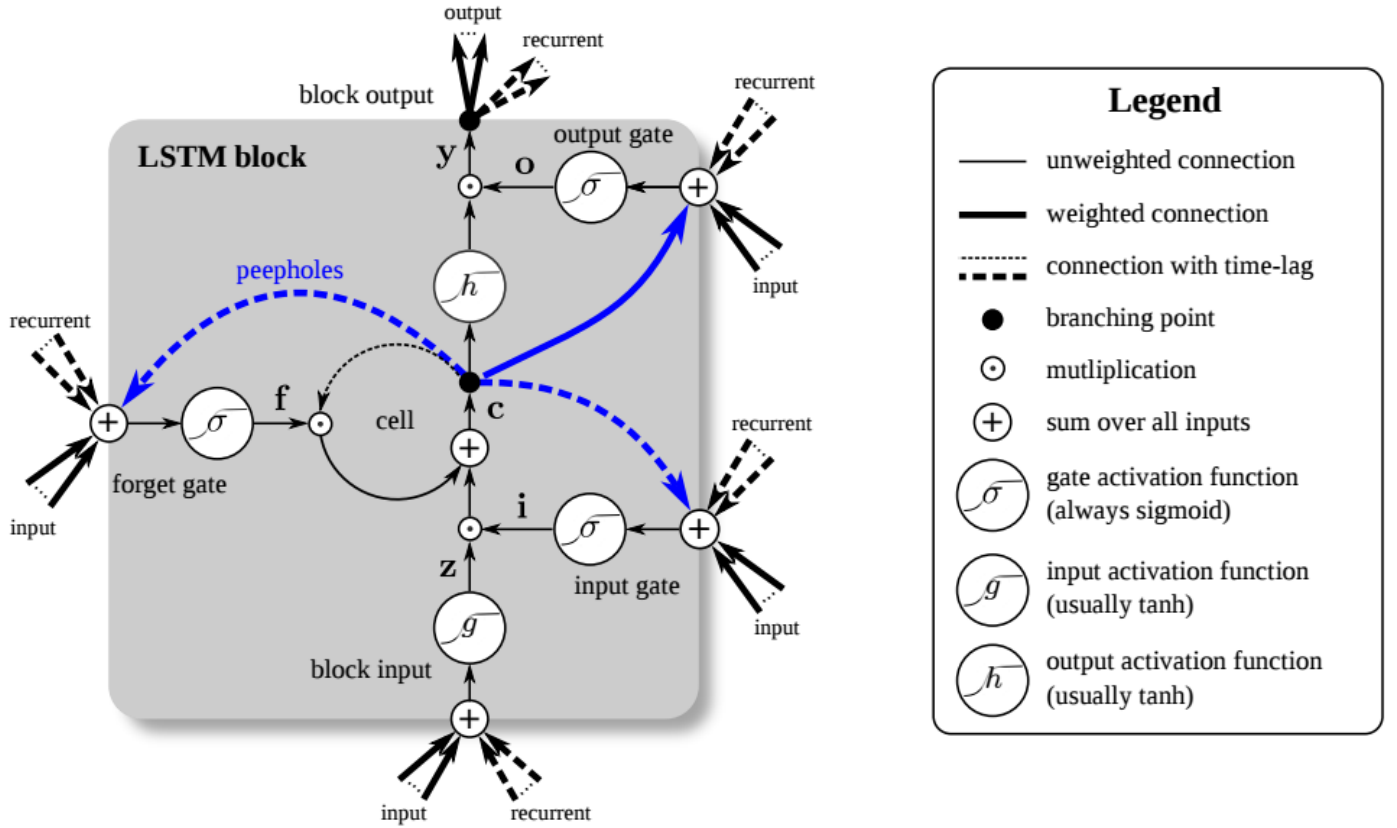
Deep Learning Neural Network



● Input Layer ● Hidden Layer ● Output Layer

Resim 4.2: Temel sinir ağlarından farklı olarak LSTM'de hidden layer'ler birbirlerine bağlıdır.

Sinir ağlarında gizli katmanlar(hidden layer) giriş ve çıkış katmanlarının arasında bulunur. Gizli katmanlar ağı giriş yapan verilerin non-linear dönüşümlerini gerçekleştirir. Katmanların büyüklüğü değişkendir. (Şekil 4.2’de düğüm yapısı gözlemlenebilir)



Şekil 4.3: Temel bir LSTM hafıza hücre modeli. LSTM’de ilk adım hücreden hangi bilginin atılacağıdır. Bu karar forget gate olarak adlandırılan sigmoid katmandan verilir. Yeni öğrenilen bir konu olduğu zaman eski konuyu unutması gerekir. Sonraki adım hangi yeni bilginin hücrede tutulacağıdır. Bu iki bölüme ayrılır. “Input Gate” katmanı hangi veriyi güncelleyeceğine karar verir. Ardından tanh katmanı bilginin hafızada tutulması için vektör yaratır. Son olarak bu iki aşama birleştirilerek “multiplication gate” ‘e gönderilir. Mevcut bilgilerle hücre hali hazırdaki hafızayı günceller. Son olarak diğer sigmoid katman(Output Gate) hangi verinin çıkış

yapacağına karar verir. Ancak yinede karar vermeden önce hafıza durumu başka tanh katmanından filtrelendir. Böylece çıkış değerleri -1 ve 0 arasında değişir.

Yinelenen sinir ağı(RNN) temel birimi değiştirilerek, computer memory-like cell olarak tasarlanması önerildi. Buda LSTM hücrelerini günümüzde var etti. (Resim 4.3)

LSTM’de temel fikir hafıza hücrelerini güncellemektir. “Memory State” yada “Cell State” olarak geçer. Hafıza hücrelerini güncellerken dikkat edilmesi gereken 2 husus mevcuttur.

- 1) Bilgi artık geçersizdir. Unutulması gerekir. (Forget Gate)
- 2) Yeni bilginin hafıza hücrelerinde tutulması gerekir. “Forget Gate buna karar verir”

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

(F.1) Sigmoid Fonksiyonu

Sigmoid fonksiyonu “0” ile “1” arasında değer alır. Forget layer, tutulan bilginin artık gerek olup olmadığını belirler. Aşağıdaki denklem bu işlemin matematiksel olarak gerçekleştirimini gösterir.

$$f_t = \sigma(W_{xf} \cdot x_t + W_{hf} \cdot h_{t-1} + b_f)$$

(F.2)

f_t forget gate'in çıkışını temsil eder. W_{xf} external input ve forget gate arasındaki bağlantı gücünü belirtir. W_{hf} önceki gizli katman ve forget gate arasındaki bağlantı gücünü belirtir. b_f forget gate'in sapmasını belirler.

Hafıza hücresinde tutulacak olan bilgiler ayrı ayrı belirlenir. Bu iki adımda gerçekleşir. Öncelikle Giriş kapısı katmanı sigmoid fonksiyonuna benzer bir işlem uygulayarak hücrede hangi bilginin tutulacağına karar verir (F.3) , İkinci olarak tanh katmanı hafıza hücresine yerleştirilmesi beklenen bilgileri liste halinde tutar. (F.4). Bu iki talimatlar birleştirilerek “multiplicative gate” ‘e gönderilir.

$$i_t = \sigma(W_{xi} \cdot x_t + W_{hi} \cdot h_{t-1} + b_i)$$

(F.3)

W_{xi} external input ve input gate arasındaki bağlantı gücünü temsil eder. W_{hi} önceki gizli katman ve input gate arasındaki bağlantı gücünü temsil eder. b_i input gate sapmasını belirtir.

$$v_t = \tanh(W_{xv} \cdot x_t + W_{hv} \cdot h_{t-1} + b_v)$$

(F.4)

Tanh , tanjant hiperbolik fonksiyonu aşağıdaki gibi tanımlanmıştır.

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

İşlemler sonucunda hafıza aşağıdaki denklem değeri olarak güncellenmiştir.

$$C_t = f_t * C_{t-1} + i_t * v_t$$

(F.5)

c_{t-1} önceki hafızadaki bilgidir.

Son olarak hücre çıkışından sağlanacak bilgidir. “Output Gate Layer” hücre girdilerini taşıyacak bir başka sigmoid fonksiyonudur. (F.5)

$$o_t = \sigma(W_{xo} \cdot x_t + W_{ho} \cdot h_{t-1} + b_o)$$

(F.6)

W_{xo} external input ve output gate arasındaki bağlantı gücünü temsil eder. W_{ho} önceki hidden layer ile output gate arasındaki bağlantı gücüdür. Hafıza hücresi h_t -1 ve 1 arasındaki değeri yollamak için tanh layer’inden geçirilir. Hücre çıkışı verisi hesaplanır. (F.7)

$$h_t = o_t * \tanh C_t$$

(F.7)

LSTM hücre bloğunda gerçekleşen tüm adımlar bu şekildedir.

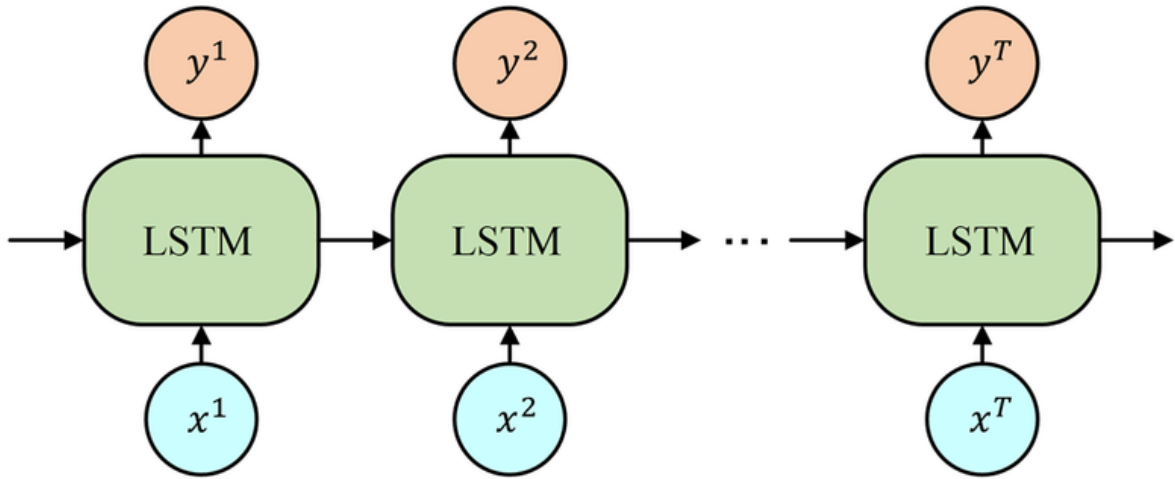
4.2 LSTM Mimarileri

LSTM için bir çok mimari bulunmaktadır. Bu mimarileri yapılan projeye göre seçmek gerekir. Her alanda farklı mimari kullanılmaktadır. Bu bölümde genellikle kullanılan mimariler anlatılmıştır.

4.2.1 Single Layer LSTM Ağı

Single Layer LSTM ağı, LSTM mimarileri arasında en basit mimaridir. N adet LSTM hücresi için tek adet hidden layer bulundurur.

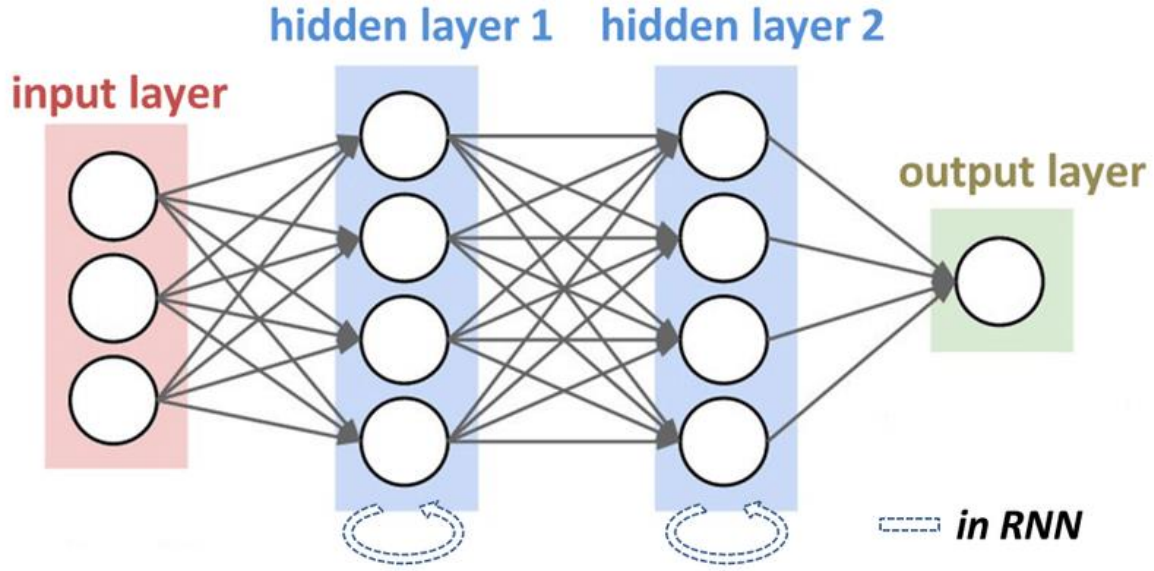
,



Resim 4.2.2.1: Tek katmanlı LSTM ağını temsil eden resim.

4.2.2 Multi Layer LSTM Ağı

Multi Layer LSTM ağı single layer ağı'nın bir parçasıdır. Aralarındaki fark ismindende anlaşılacağı üzere multi layer lstm ağı birden fazla hidden layer içerir.

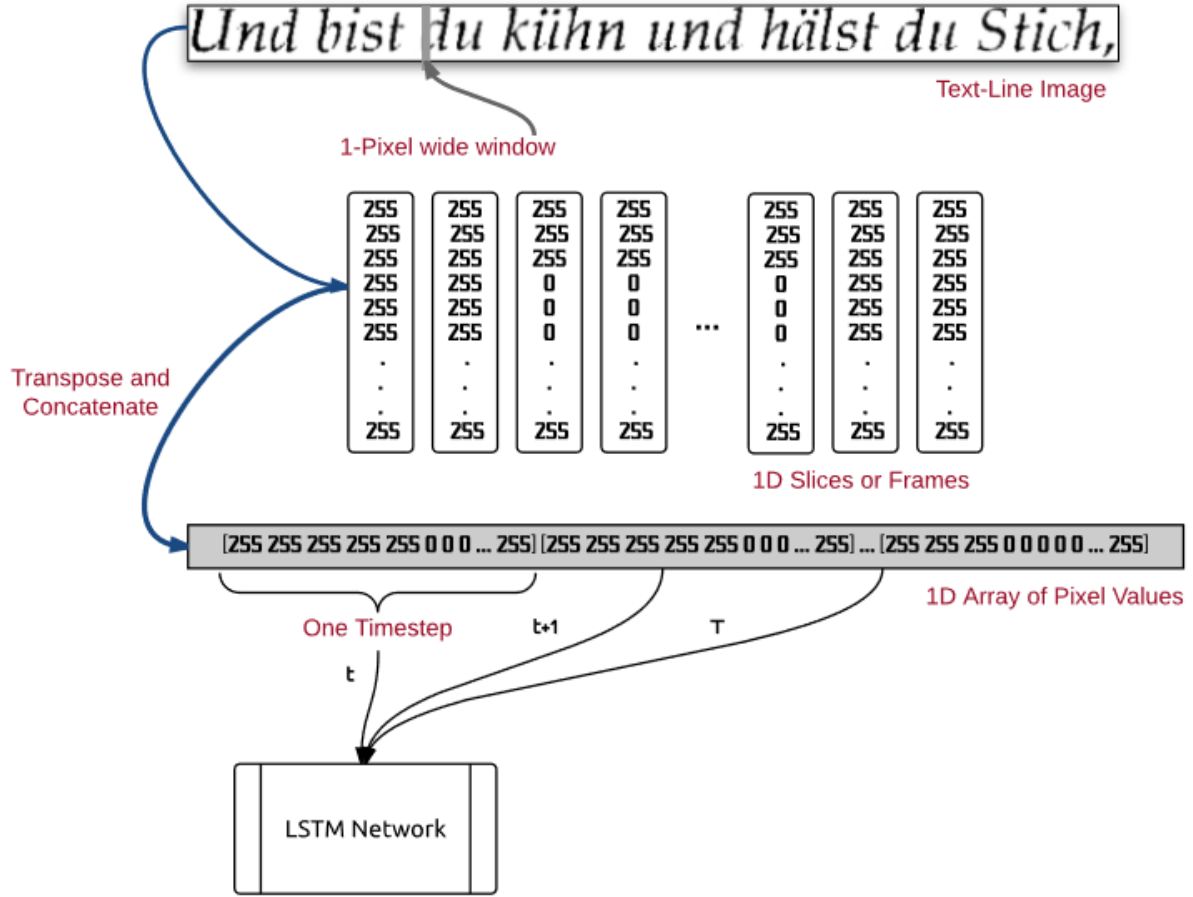


Resim 4.2.2.2: Çok katmanlı LSTM ağını temsil eden resim.

4.2.3 One Dimensional LSTM(1D-LSTM) Ağı

One dimensional LSTM(1D-LSTM) ağında, giriş dizisi 1 piksel genişliğinde ve görüntü yüksekliğine eşit yükseklikte bir kayan pencere tarafından geçilir. Bu giriş dizisini tek boyutlu bir diziye dönüştürür. Görüntünün yüksekliği, 1D dizinin

derinliđi olarak adlandırılır ve bu řekilde elde edilen grntnn her dilimine “ereve” denir. řekil 4.2.2.3 ‘de grnty tek boyutlu diziye dnřtrme iřlemi grlmektedir.



řekil 4.2.2.3: Image dosyasını tek boyutlu diziye dnřtrr.

Tesseract-OCR ile yapacađımız training methodunda One dimensional LSTM mimarisi kullanılır.

4.2.4 Parametreler –(Ek Chapter)

LSTM ağlarının belirli bir problem üzerinde tatmin edici bir şekilde performans göstermesi için ayarlanması gereken çeşitli parametreler vardır. Bu bölümde projede gerçekleştirilen one dimensional LSTM parametrelerine değineceğiz.

1D-LSTM ağları için, en önemli parametre hidden-layer büyüklüğüdür. Yani gizli katmandaki LSTM hücrelerinin sayısıdır. Hücre sayısını artırmak performansı artırır. Ancak aynı zamanda training sürecinide uzatır. Diğer parametreler learning rate ve momentumdur. LSTM performansı genelde bu parametrelerin değiştirilmesinden etkilenmez (Learning rate ve momentum için)

5 Tesseract-OCR

Tesseract 4.0, gerekli bilgi işlem gücünde önemli bir artış sağlayarak önceki sürümlere göre önemli ölçüde daha yüksek doğruluk sunan yeni bir sinir ağı tabanlı karakter tanıma motoru içerir. Karmaşık dillerde temel Tesseract'dan daha hızlıdır.

Sinir ağları, önemli ölçüde veri seti gerektirir ve temel tesseract'tan çok daha yavaş eğitilir. Latin tabanlı diller için, sağlanan model verileri, yaklaşık 4500 yazı tipini kapsayan 400000 metin satırı üzerinde eğitilmiştir. Tesseract 4.0 spesifik dil dosyaları için eğitilme süresi birkaç günden birkaç haftaya kadar uzanabilir. Eğer

oluşturduğunuz eğitim seti eğer yetersizse tekrardan eğitebilirsiniz. Ne kadar fazla veri seti ve öğrenmek için zaman harcanırsa o kadar yüksek doğruluk oranı sunar.

Eğitim için birden fazla seçenek vardır:

- Fine tune. Tesseract-OCR'ın github kaynağı üzerinde önceden çalışılmış ve üretilmiş olan spesifik dil dosyaları bulunur. Bu mevcut eğitim verileri için kendi verisetinizi kullanarak eğitebilirsiniz. Daha önceden geliştirildiği için az miktarda veri seti ile bile çalışabilir
- Sıfırdan yeniden eğitmek, çok büyük sayıda veriseti gerekir. Her ne kadar spesifik eğitim modelleri için başarı gösterebilir. Eğitilen modeller haricinde karakter okumada başarısız olacaktır.

Yukarıdaki seçenekler kulağa farklı gelse de, temel olarak eğitim adımları aynıdır. Bir sonraki bölümde eğitim adımlarına geçeceğiz.

Tesseract 4.0 versiyonunda eski tanıma motoru hala mevcuttur ve eğitilebilir halde bulunur. Ancak gelecek versiyonlarda gerek duyulmadığı takdirde silinebilir.

5.1 Tesseract 4.0 Genel Bakış

Tesseract 4.0 yeni sinir ağı alt sistemi içerdiğinden söz etmiştik. Bu sinir ağı OCRopus'un python tabanlı LSTM uygulamasına dayanmaktadır. Ancak C++'da Tesseract için yeniden dizayn edilmiştir. Tesseract'daki sinir ağı sistemi TensorFlow'dan eskidir, ancak TensorFlow için de kullanılabilen Değişken Grafik Spesifikasyon Dili(VGSL) adlı bir ağı tanımlama dili olduğu için onunla uyumludur.

VGSL'in mantığı, çok fazla öğrenim gerçekleşmeden bir sinir ağı kurmanın ve onu eğitmenin mümkün olmasıdır. Python, TensorFlow öğrenmeye hatta herhangi bir C++ kodu yazmaya gerek yoktur. Yalnızca sözdilimsel olarak doğru ağ tanımları oluşturmak için VGSL belirtim dilini yeterince iyi anlamak gerekir.

5.1.1 Tesseract 4.0 İle Entegrasyon

Tesseract 4.0 sinir ağı alt sistemi, çizgi tanıyıcı olarak Tesseract'a entegre edilmiştir. Büyük bir metni tanımak için mevcut düzen analizi ile kullanılabilir. Sinir ağı 4.0 ile varsayılan olarak gelmektedir. Tek bir yazıyı tanımak için `-psm 13` komutu kullanılabilir.

5.1.2 Sistem Gereksinimleri

Tesseract 4.0 sinir ağı alt sistemi, İşlemciyi yoğun olarak hesaplama işlerinde kullanır.

- SSE ve/veya AVX'I destekleyen Intel/AMD işlemciler, çekirdek matris çarpımlarının SIMD paralelleştirilmesinden yararlanır.

Birçok çekirdeğe ve AVX'e sahip bir makinede, kolay bir ingilizce yazılı bir resim dosyasında, gerçek zamanlı olarak iki kat daha fazla zaman alır. Temel tesseract 7 kat CPU kullanırken Hintçe, Temel tesseract'tan daha fazla CPU kullanır.

5.2 Tesseract Training

Bu bölümde projenin kapsamında olan 204 adet Captcha verisetini kullanarak yapay öğrenme gerçekleştireceğiz. Sonucunda sıfırdan öğrettiğimiz modeli hazır olan Tesseract dilleri ile karşılaştıracacağız. Yaptığımız öğrenme sonucunda başarılı olup olmadığımızı gözlemleyeceğiz. Adım adım gereksinimleri ve nasıl bir yoldan gideceğiz açıklayacağız. Aşağıdaki resimde klasörde bulunan ve training edeceğimiz dataset örneklerini görebilirsiniz.

Training sonucunda 1042 adet captcha üzerinden Accuracy testi yapacağız.



Resim5.1: Resimdeki image dosyaları elimizde bulunan 204 adet datasetinden örnekleri gösterir.

5.2.1 Gereksinimler

Training işlemine başlamadan önce gerekli olan yazılımları listersek;

- 1) Linux tabanlı işletim sistemi
- 2) Training edilecek veriseti (Resim 5.1)
- 3) Tesseract-OCR
- 4) Libicu-dev,libpango1.0-dev,libcairo2-dev Kütüphaneleri
- 5) Fine Tuning öğrenme için, LSTM için training edilmiş tessdata_best -> eng.traineddata modeli (Öğrenmeye buradan başlayacak ve öğrenme sürecini çok kısaltacak)

4.2.2 Adımlar

Training işlemine başlamadan önce lstmtraining programının command-line opsiyonlarındaki bizim için gerekli olan parametreleri inceleyelim.

<code>learning_rate</code>	<code>double</code>	<code>10e-4</code>	Initial learning rate for SGD algorithm.
----------------------------	---------------------	--------------------	--

Learning_rate train işlemi gerçekleştirilirken eğer kendimiz belirlemezsek default olarak 10e-4 sayısını baz alarak training eder.

momentum	double	0.5	Momentum for alpha smoothing gradients.
----------	--------	-----	---

Momentum default olarak 0.5 baz alınır.

adam_beta	double	0.999	Smoothing factor squared gradients in ADAM algorithm.
-----------	--------	-------	---

Adam algoritma default olarak 0.999 baz alınır.

net_mode	int	192	Flags from <code>NetworkFlags</code> in <code>network.h</code> . Possible values: 128 for Adam optimization instead of momentum; 64 to allow different layers to have their own learning rates, discovered automatically.
----------	-----	-----	---

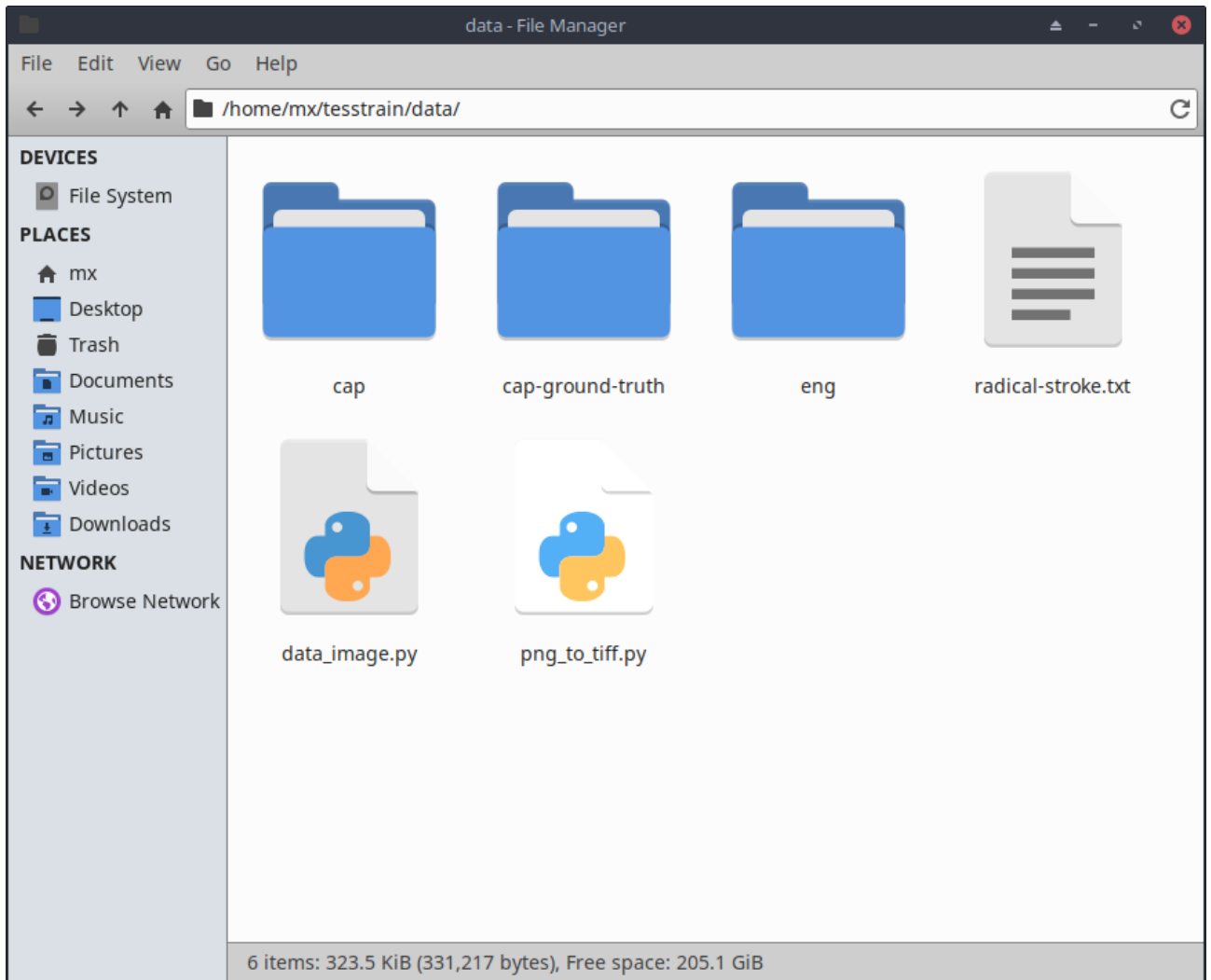
Net_mode flagı için 2 opsiyon bulunur. Parametre eğer 128 seçilirse momentum yerine Adam optimizasyonu kullanılır. 64 için farklı layerler kendi learning_rate'lerine sahip olur. Default için otomatik keşfedilir.

Bu parametreler girilmediği takdirde default olarak verilen değerlere göre training işlemi gerçekleştirilir.

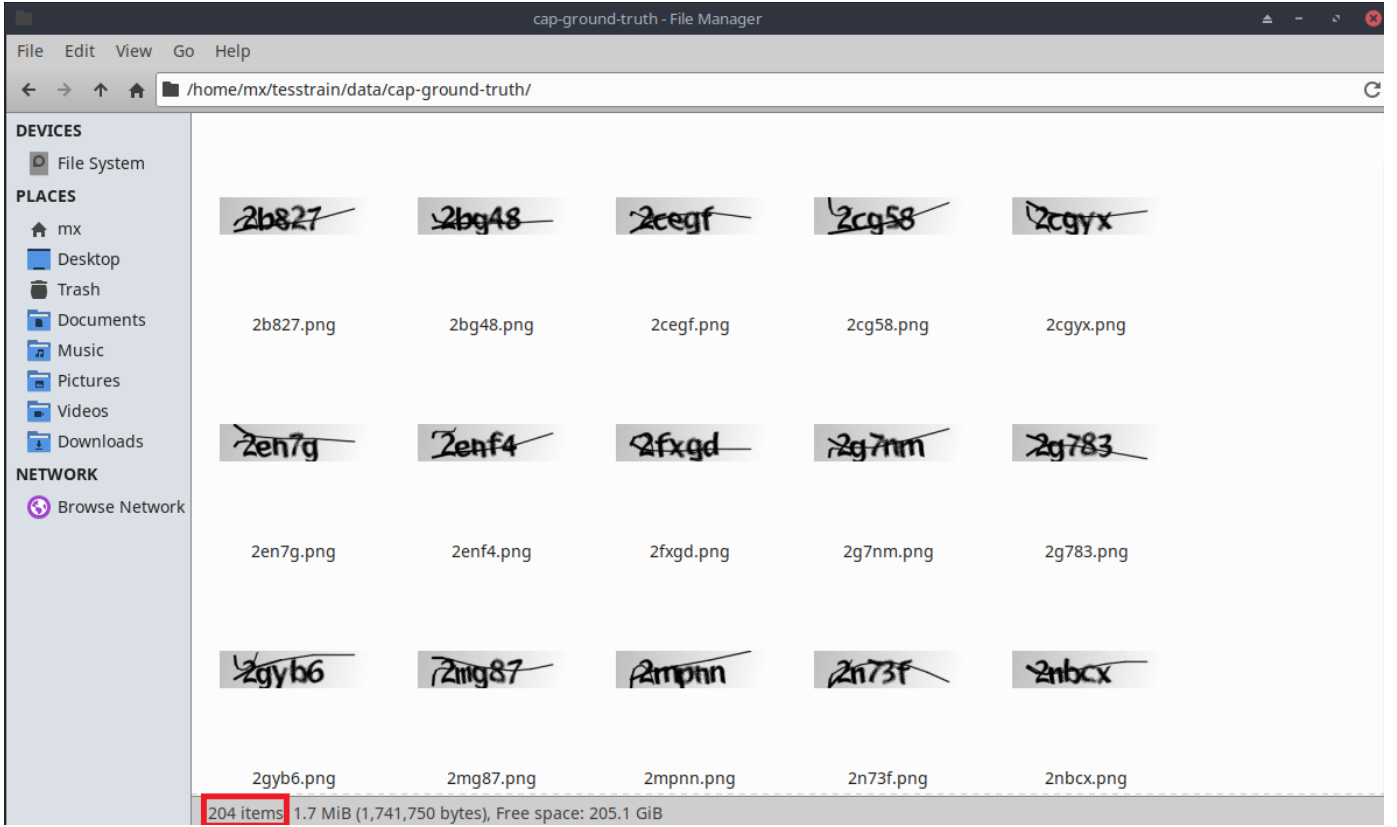
Traininge başlamadan önce tesseract'ın training için oluşturduğu dosyaları github repository'sinden klonluyoruz <https://github.com/tesseract-ocr/tesstrain>

Training edeceğimiz datasetlerini klonladığımız klasör dizindeki data yoluna **data/MODEL_NAME-ground-truth** adında klasör açıyoruz. MODEL_NAME bizim oluşturacağımız dil dosyasının ismini ifade eder. Örneğin ben cap olarak model oluşturacağım.

Yani **data/cap-ground-truth** klasörünü oluşturdum.



Resimde görüldüğü gibi data yoluna cap-ground-truth adında klasör oluşturduk. Bu dosyanın içine 204 adet verisetimizi atalım.



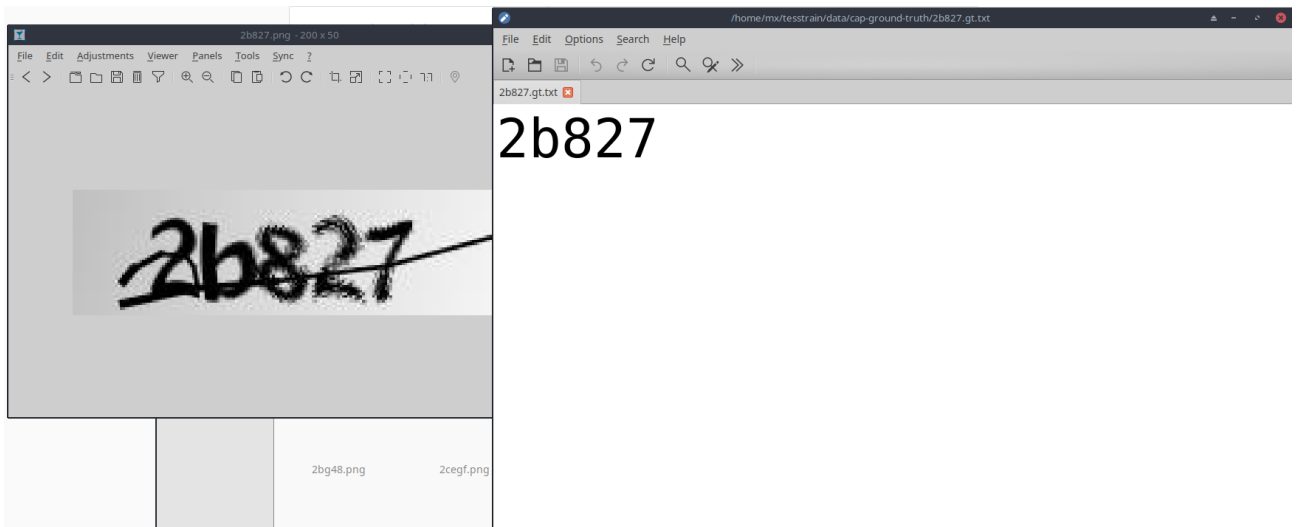
204 adet verisetini klasöre attıktan sonra her işlenecek resim için resimde görülen tek satırlık yazıları (2b827,2bg48,2cegf ...) her resim için “.gt.txt” uzantısında text dosyası oluşturarak bu text dosyasının içine resimde gözüken yazı bulundurması gerekiyor. Bunun için ufak bir python kodu yazarak bütün verisetleri için bu işlemi gerçekleştirdim.

```
File Edit Options Search Help
data_image.py x
import os

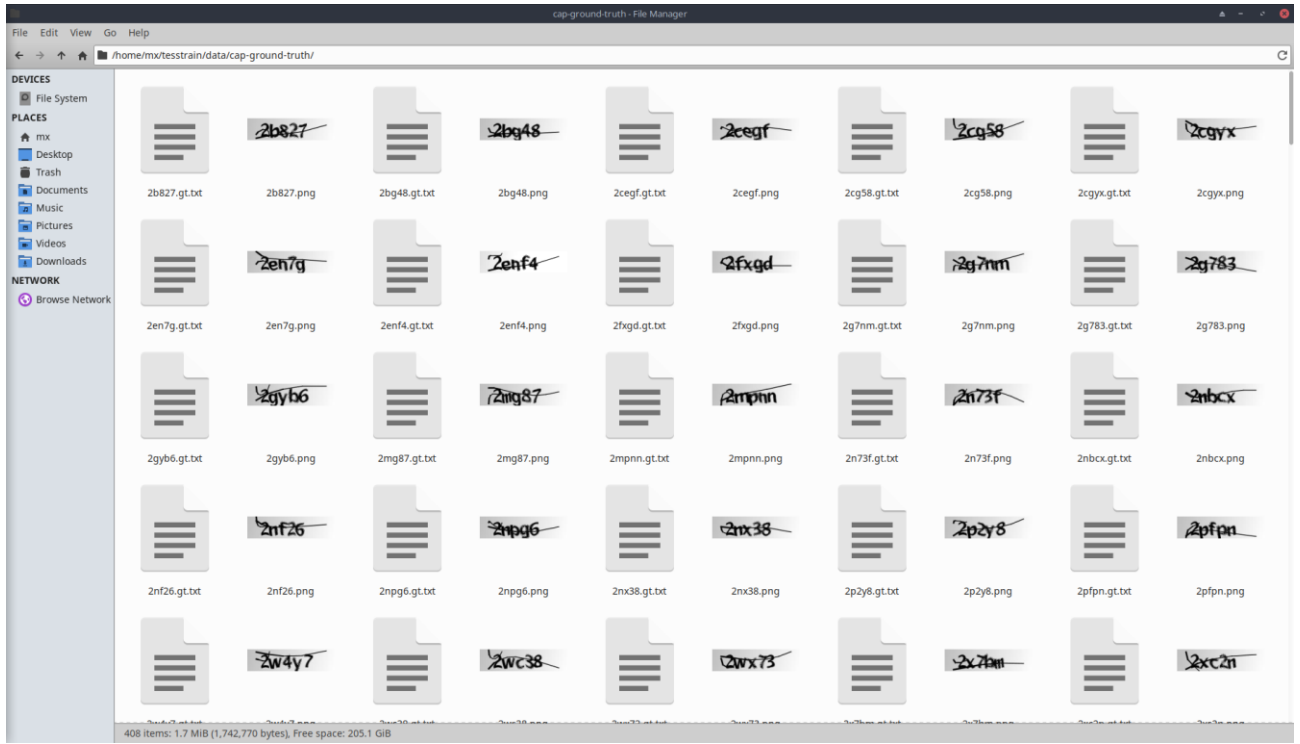
for x in os.listdir("./cap-ground-truth"):
    image_name = x.split(".")[0]
    f0 = open("./cap-ground-truth/"+image_name+".gt.txt", "w")
    f0.write(image_name)
    f0.close()
```

Bu python kodu verisetinin bulunduğu klasördeki her resim için örneğin “2b287.png” captchası için “2b287.gt.txt” dosyası oluştururak bu dosyaya “2b287” yazıyor.

Python kodunu çalıştırdıktan sonra

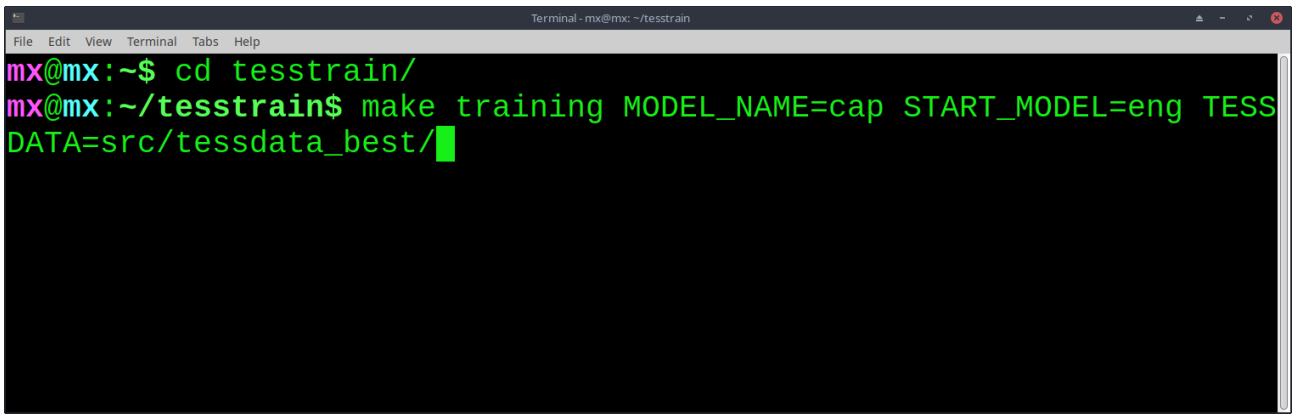


Her dosya için bu işlem tamamlanmış olacaktır.



Artık training işlemine geçebiliriz.

Linux terminalini açarak şu komutu satırını kullandık. “make training
MODEL_NAME=cap START_MODEL=eng TESSDATA=src/tessdata_best” bu
komut satırını sırasıyla açıklayalım



```
mx@mx:~$ cd tesstrain/  
mx@mx:~/tesstrain$ make training MODEL_NAME=cap START_MODEL=eng TESS  
DATA=src/tessdata_best/
```

Make training: make training ile training işlemini gerçekleştireceğimiz ilk komut, bu komuttan sonra gerekli parametreleri girmemiz gerek.

MODEL_NAME, model ismi yani data klasöründe oluşturduğumuz dil dosyasının ismi cap olarak seçtiğim için cap yazıyoruz.

START_MODEL, burada fine tuning öğrenme methodunu gerçekleştirdiğimiz için yani öğrenmeye belirli bir noktadan, dilden başlayacağını gösteriyoruz. Eng yani english dil dosyası ile başlıyoruz.

TESSDATA, burada LSTM training için geliştirilmiş dil dosyalarının bulunduğu konumu yazıyoruz. Github'dan clone ettiğimiz tessdata_best klasörünü işaret eder. /src/tessdata_best klasöründe bulunuyor.

Sonunda öğrenme işlemine başladığımızda her veriseti için(204 resim) öğrenme gerçekleşmeye başlıyor. Resim(4.2.2.6)

LSTM parametrelerini girmedik. Default olarak belirli değerler üzerinden training yapacağız

```
Terminal - mx@mx: ~/tesstrain
File Edit View Terminal Tabs Help
Loaded file data/cap/checkpoints/cap_checkpoint, unpacking...
Successfully restored trainer from data/cap/checkpoints/cap_checkpoint
Loaded 1/1 pages (1-1) of document data/cap-ground-truth/6ge3p.lstmf
Loaded 1/1 pages (1-1) of document data/cap-ground-truth/5wddw.lstmf
Loaded 1/1 pages (1-1) of document data/cap-ground-truth/3bx86.lstmf
Loaded 1/1 pages (1-1) of document data/cap-ground-truth/7p852.lstmf
Loaded 1/1 pages (1-1) of document data/cap-ground-truth/2wx73.lstmf
Loaded 1/1 pages (1-1) of document data/cap-ground-truth/3p4nn.lstmf
Loaded 1/1 pages (1-1) of document data/cap-ground-truth/6b4w6.lstmf
Loaded 1/1 pages (1-1) of document data/cap-ground-truth/7gnge.lstmf
Loaded 1/1 pages (1-1) of document data/cap-ground-truth/6wb76.lstmf
Loaded 1/1 pages (1-1) of document data/cap-ground-truth/6n5fd.lstmf
Loaded 1/1 pages (1-1) of document data/cap-ground-truth/3nw7w.lstmf
At iteration 349/10000/10000, Mean rms=0.133%, delta=0.177%, char train=0.56%, word
train=1.8%, skip ratio=0%, New worst char error = 0.56 wrote checkpoint.

Finished! Error rate = 0.1
lstmtraining \
--stop_training \
--continue_from data/cap/checkpoints/cap_checkpoint \
--traineddata data/cap/cap.traineddata \
--model_output data/cap/cap.traineddata
Loaded file data/cap/checkpoints/cap_checkpoint, unpacking...
mx@mx:~/tesstrain$
```

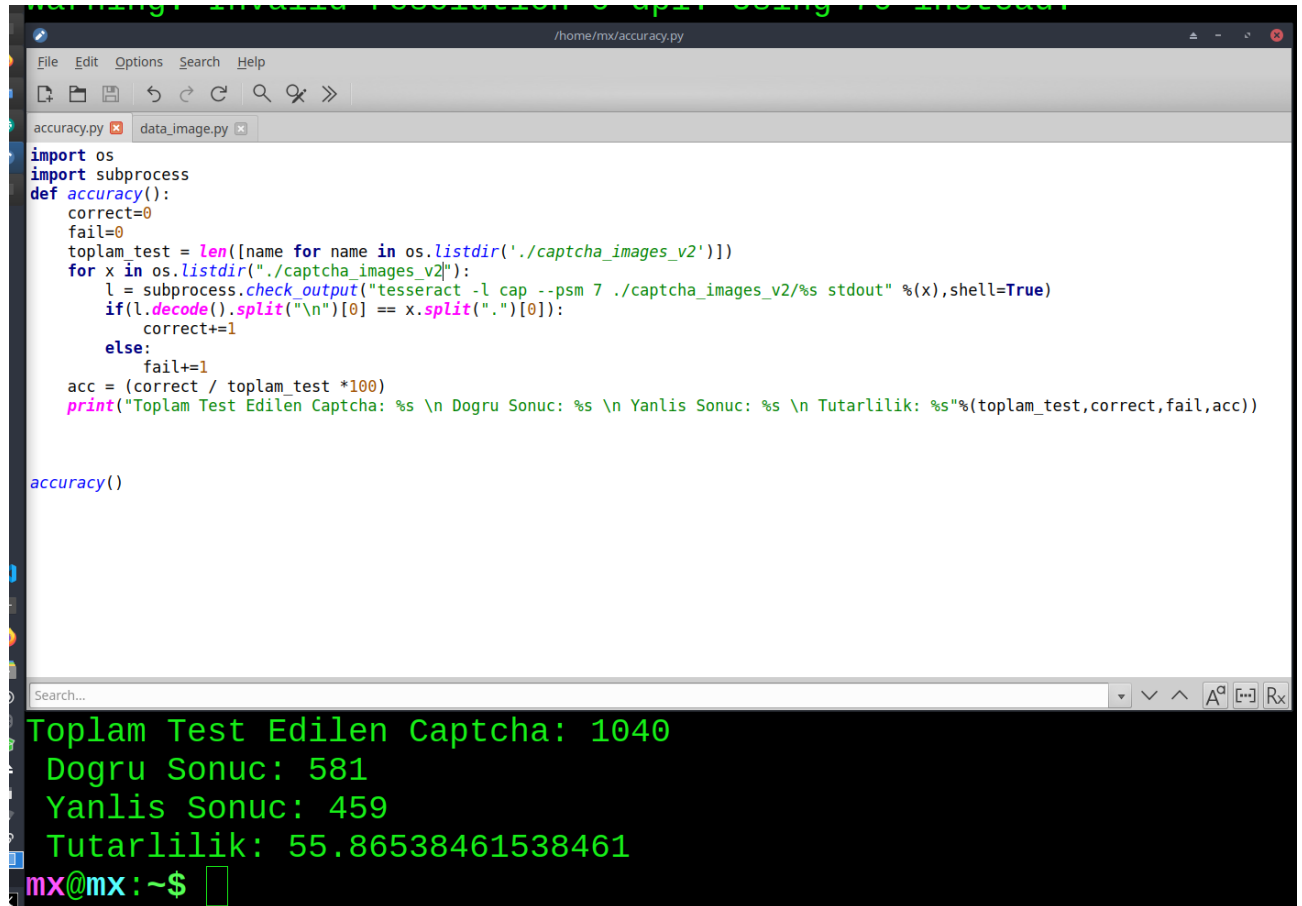
Yaklaşık 30 dakikadan sonra Hata oranı=0.1'e düştü. Dil dosyası oluşturuldu ve /data/cap/cap.traineddata yoluna dil dosyası yazdırıldı.

Cap.traineddata'yı tesseract'ın kurulu olduğu dizin yani linux'de /usr/local/share/tesseract4.0 konumundaki dil dosyalarının bulunduğu bölüme tessdata klasörüne kopyaladık.

Bir sonraki bölümde dil dosyasını test edeceğiz.

4.2.3 Sonuç

Train ettiğimiz 204 adet veriseti yaklaşık 30 dakika sonra sonlandı. Şimdi bu oluşturduğumuz dil için ufak bir python algoritmasıyla Accuracy testi yapalım. 1000 adet captcha testinden % kaç başarılı olacak görelim.



```
Warning: Invalid fontset definition is being used.
/home/mx/accuracy.py
File Edit Options Search Help
accuracy.py data_image.py
import os
import subprocess
def accuracy():
    correct=0
    fail=0
    toplam_test = len([name for name in os.listdir('./captcha_images_v2')])
    for x in os.listdir("./captcha_images_v2"):
        l = subprocess.check_output("tesseract -l cap --psm 7 ./captcha_images_v2/%s stdout" %x,shell=True)
        if(l.decode().split("\n")[0] == x.split(".")[0]):
            correct+=1
        else:
            fail+=1
    acc = (correct / toplam_test *100)
    print("Toplam Test Edilen Captcha: %s \n Dogru Sonuc: %s \n Yanlis Sonuc: %s \n Tutarlilik: %s"%(toplam_test,correct,fail,acc))

accuracy()

Toplam Test Edilen Captcha: 1040
Dogru Sonuc: 581
Yanlis Sonuc: 459
Tutarlilik: 55.86538461538461
mx@mx: ~$
```

204 resimle training sonucunda, 1040 adet resim denemesinde 581 doğru sonuç 459 yanlış sonuç verdi ve tutarlılık %55 olarak çıktı.

Tesseract 4.0 ile resimlere herhangi bir pre-processing yapmadan 204 adet verisetiyle fine-tuning öğrenme sonucunda ortalama bir sonuç aldık. Elde ettiğimiz bu başarıdan yazı tabanlı captcha'nın günümüzde neredeyse hiç kod yazmadan kolaylıkla manipüle edilebileceğini gördük. Dolayısıyla yazılım geliştiricilerinin kırması daha zor olan resim tabanlı captcha'ları sistemlerinde kullanmalarını tavsiye ederim.

KAYNAKLAR

- [1] Tesseract-OCR dosya kaynakları <https://github.com/tesseract-ocr/tesseract> (4.10.2021 Erişildi)
- [2] Tesseract4.0 Nöral Öğrenme talimatları <https://tesseract-ocr.github.io/tessdoc/tess4/NeuralNetsInTesseract4.00> (5.27.2021 Erişildi)
- [3] Tesseract 4.0 Training Prosedürü <https://tesseract-ocr.github.io/tessdoc/tess4/TrainingTesseract-4.00.html> (4.12.2021 Erişildi)
- [4] Verisetleri Kaynağı <https://www.kaggle.com/fournierp/captcha-version-2-images> (4.10.2021 Erişildi)
- [5] Manuel HUBER,”CAPTCHA Recognition With Adaptable Neural Networks”, Yüksek Lisans Tezi, *Hochschule LandshutFakultät Informatik*,Almanya,2019
- [6] Karmand Hussein ABDALLA “AN ANALYSIS OF THE STRENGTH OF CAPTCHA USED BY SOME TURKISH UNIVERSITIES TO PROTECT STUDENT INFORMATION”,*Fırat Üniversitesi*,Türkiye,Temmuz,2017
- [7] LSTM(Long-Short-Term-Memory) bilgi kaynağı https://tr.wikipedia.org/wiki/Long_short-term_memory
- [8] Adnan Ul-Hasan “Generic Text Recognition usingLong Short-Term Memory Networks”, Doktora Tezi, *Kaiserslautern Üniversitesi*,Almanya,2016
- [9] LSTM Çalışma Prensipi <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- [10] Cem Direkoğlu,Melike Şah Direkoğlu, “Worldwide and Regional Forecasting of Coronavirus (Covid-19) Spread using a Deep Learning Model”, <https://www.medrxiv.org/content/10.1101/2020.05.23.20111039v1.full.pdf>