RĪGAS TEHNISKĀ UNIVERSITĀTE

Datorzinātnes un informācijas tehnoloģijas fakultāte

Datorvadības, automātikas un datortehnikas institūts

Attēlu apstrādes un datorgrafikas katedra

Priekšmeta “Attēlu apstrāde un analīze”

**Kursa darbs**

“Uz neironu tīkla balstīta optiska rakstzīmju atpazīšanas programatūras izstrāde”

**Darba autores:** Anete Brenčuka

Evija Cibuļska

Maija Iveta Gaiduka

Rīga 2019

# Saturs

# 

[Ieskats 2](#_Toc26901200)

[Izmantotie avoti 2](#_Toc26901204)

[Ievads 4](#_Toc26901205)

[1. Analizējamā attēla pārveidošana 5](#_Toc26901206)

[2. OpenCV 5](#_Toc26901207)

[3. Pielietotās attēlu apstrādes metodes 5](#_Toc26901208)

[3.1 Attēla pārveidošana par melnbaltu 6](#_Toc26901209)

[3.2 Attēla asināšana un Laplasa tests miglainības noteikšanai 6](#_Toc26901210)

[3.2.1 Attēla asināšana 8](#_Toc26901211)

[3.4. Attēla kontrasta un spilgtuma korekcija 10](#_Toc26901212)

[3.5. Attēla attīrīšana no trokšņiem 11](#_Toc26901213)

[3.6 Attēlā esošā teksta pagriešana 15](#_Toc26901214)

[4. Neironu tīklu apmācība burtu atpazīšanai 20](#_Toc26901215)

[4.1. OCR Stratēģija 20](#_Toc26901216)

[4.2 Apmācības datu sagatave 21](#_Toc26901217)

[4.3. Tesseract apmācība izmantojot sagatavotos datus 22](#_Toc26901218)

[5. Lietotnes GUI izveide 28](#_Toc26901219)

[5.1 QT designer programmatūra 28](#_Toc26901220)

[5.2 Step by step installation and design process 28](#_Toc26901221)

[5.3 Python kodu izveide un dizaina integrācija 29](#_Toc26901222)

[5.4 Nepieciešamās bibliotēkas 31](#_Toc26901223)

[6. Apmācītā neironu tīkla izmantošana rakstīta teksta atpazīšanai 32](#_Toc26901224)

# 

# Ieskats

### **Projekta mērķis:**

Projekta mērķis ir izveidot sistēmu, kas spēj atpazīt tekstu attēlā latviešu un angļu valodās.

### **Mērķauditorija:**

Dokumenta mēŗķauditorija ir projektā ieinteresētās personas.

### **Definīcijas un apzīmējumi:**

Zemāk redzama tabula ar dokumentā izmantotajām definīcijām un apzīmējumiem.

|  |  |
| --- | --- |
| **Definīcija/ apzīmējums** | **Paskaidrojums** |
| UNTBORAP [UN-Tē-BŌRAP] | Uz neironu tīkla balstīta optiska rakstzīmju atpazīšanas programatūras |
| OCR | Optiska simbolu atpazīšana (Optical Character Recognition) |
| cv2 | opencv-python bibliotēka |

# Izmantotie avoti

<https://docs.opencv.org/3.4/d5/d69/tutorial_py_non_local_means.html>;

<https://arxiv.org/ftp/arxiv/papers/1505/1505.03489.pdf>;

<https://improvephotography.com/9195/digital-noise-in-photography/>;

<https://scialert.net/fulltextmobile/?doi=jas.2014.950.954>;

<https://www.ripublication.com/irph/ijece/ijecev5n1__3.pdf>;

<https://www.researchgate.net/publication/324476545_A_Study_of_the_Effects_of_Gaussian_Noise_on_Image_Features>;

<https://stackoverflow.com/questions/2440504/noise-estimation-noise-measurement-in-image>;

<https://www.deepdyve.com/lp/elsevier/fast-noise-variance-estimation-mOpgmPcOaA>;

<https://www.pyimagesearch.com/2015/09/07/blur-detection-with-opencv/>;

<https://medium.com/@almutawakel.ali/opencv-filters-arithmetic-operations-2f4ff236d6aa>;

<https://web.stanford.edu/class/cs448f/lectures/2.1/Sharpening.pdf>;

<https://www.researchgate.net/figure/Comparison-of-different-Unsharp-masking-approaches-for-Image-sharpening_tbl1_305985620>;

<https://www.researchgate.net/publication/305985620_A_Review_on_the_Image_Sharpening_Algorithms_Using_Unsharp_Masking>;

<https://docs.opencv.org/3.4.3/d3/dc1/tutorial_basic_linear_transform.html>;

<https://techtutorialsx.com/2018/06/02/python-opencv-converting-an-image-to-gray-scale/>

<https://arxiv.org/pdf/1707.02051.pdf>;

Python Skew Image:

[1.] <https://www.pyimagesearch.com/2017/02/20/text-skew-correction-opencv-python/>

**Sources TESSERACT:**

[1] <https://github.com/tesseract-ocr/tesseract/wiki/Compiling>

[2] <https://guides.library.illinois.edu/c.php?g=347520&p=4121426>

[3] <https://docs.docker.com/engine/reference/builder/>

[4] <https://github.com/tesseract-ocr/>

[5] <https://packages.ubuntu.com/bionic/tesseract-ocr-all>

[6] <https://github.com/tesseract-ocr/tesseract/wiki/Data-Files>

[7] [https://github.com/tesseract-ocr/tesseract/issues/2357](https://github.com/tesseract-ocr/tesseract/issues/2357#issuecomment-477239316)

OpenCV auto adjust contrast & brightness:

[1] <https://answers.opencv.org/question/75510/how-to-make-auto-adjustmentsbrightness-and-contrast-for-image-android-opencv-image-correction/>

[2] <https://answers.opencv.org/question/194297/auto-adjusting-contrast-and-brightness-with-opencv-python/>

[1]Python Skew Image: <https://www.pyimagesearch.com/2017/02/20/text-skew-correction-opencv-python/>

QtDesigner:

**[]** <https://doc.qt.io/qt-5/qtdesigner-manual.html>

OPENCV:

<https://opencv.org/about/>

# Ievads

Laikā, kad aizvien vairāk ikdienas pamata procesi tiek digitalizēti un piemēroti dažādu iekārtu izpildei, rodas jautājums: Kas notiek ar jau esošo informāciju un kā to iespējams efektīvi saglabāt un pārveidot digitalizētajai ikdienai atbilstošā formātā? Pietam, viens no visgrūtāk pārnesamajiem formātiem ir ar roku rakstītie dokumenti vai cita veida raksti. Parasti vienīgais veids, kā pārcelt šos papīrus uz datorizētu vidi ir dublicēt visu informāciju, ar rokām pārrakstot to datorrakstā.

Iesaistīt cilvēku kā informācijas nodevēju starp fizisko papīru un datoru, savukārt ne vienmēr ir iespējams, piemēram, ja tiek apskatīts liels kvantums ar roku rakstītās informācijas, un tas ne vienmēr garantē 100% precīzu informācijas pārnesi.

Tapēc nepieciešams izstrādāt sistēmu, kas ir spējīga uztvert un atpazīt ar roku rakstītu tekstu un korekti pārvērt to datorrakstā. Šāda sistēma, ne tikai atvieglos darbu cilvēkam, kam būtu informācija jāpārraksta manuāli, bet arī efektīvi samazinās informācijas pārveides laiku un resursus. Darba ietvaros tika izveidots šādas sistēmas prototips, Uz neironu tīkla balstīta optiska rakstzīmju atpazīšanas programma (turpmāk tekstā - UNTBORAP), kas tika trenēta atpazīt, ar roku rakstītus latviešu valodas burtus un uzlabot attēla kvalitāti, paredzot gadījumus, kad sistēmas lietotājs nav ievadījis optimāli kvalitatīvu attēlu.

UTBORAP trenēšanā tika izmantota neironu tīkla bibliotēka Tesseract un vairāku rakstīšanas stilu veidos ar roku sagatavoti burtu piemēri.

# Analizējamā attēla pārveidošana

Lietotājs, kas ievada UNTBORAP sistēmā attēlu nav spējīgs visos gadījumos nodrošināt datorredzei pieņemamu attēla kvalitāti, tāpēc ir nepieciešams ievadīto attēlu dažādos veidos pārbaudīt un, ja nepieciešams, pārveidot, izmantojot dažādas attēla apstrādes metodes. Sistēmai ir jābūt spējīgai noteikt to, vai ievadītajam attēlam ir nepieciešams pielietot konkrētas apstrādes metodes, jo pretējā gadījumā tiks sabojāts kvalitatīvs, OCR apstrādei atbilstošs attēls.

# 2. OpenCV

Darba laikā tika izvēlēts darboties ar OpenCV (Open Source Computer Vision Library) kas ir atvērta pirmkoda datorredzes un mašīnapmācības programmatūras bibliotēka. OpenCV tika izveidota lai nodrošinātu vienotu infrastruktūru datorredzes lietotnēm un lai paātrinātu mašīn uztveres izmantošanu komerciāli izmantojamos produktos.

Bibliotēka satur vairāk kā 2500 optimizētis algoritmus, kas aptver gan klasisku, gan arī pašu jaunāko datorredzes un mašīnapmācības algoritmu kopas. Šos algoritmus var izmantot lai atrastu un atpazītu sejas, identificētu dažādus objektus, klasificētu cilvēku kustību, kas ierakstīta video formātā, izsekotu kameru kustībām, izgūtu objektu 3D modeļus, ražotu 3D punktu mākoņus no stereo kamerām, sašūtu kopā attēlus lai tie rezultētos augstas definīcijas attēlā, kas atbilst scēnai, atrastu līdzīgus attēlus attēlu datubāzē, attīrītu sarkano acu efektu no attēliem, kas uzņemti izmantojot zibspuldzi, sekotu acu kustībām, atpazītu ainavas un nodrošinātu marķierus, kurus pielietot pārklājot augmentēto realitāti, utt.

Tā izmanto C++, Python, Java un MATLAB saskarnes un atbalsta Windows, Linux, [Android](https://opencv.org/android/) un Mac operētājsistēmas. OpenCV sliecas vairāk uz reālā laika lietotņu izstrādi. Eksistē vairāk kā 500 algoritmi un apmēram 10 reižu vairāk funkciju, kas sastāda un atbalsta šos algoritmus. OpenCV pirmkods pamatā ir rakstīts C++ valodā.

## 3. Pielietotās attēlu apstrādes metodes

UNTBORAP sistēmā automātiski tiek pielietotas šādas metodes:

* [Attēla pārveidošana par melnbaltu attēlu;](#_roie05xgnpac)
* [Attēla asināšana;](#_pzgzl0l3rnnw)
* [Attēla fona dzēšana;](#_oqqudt3k0o1h)
* [Attēla kontrasta un gaišuma maiņa;](#_rc0z5wf0s6ln)
* [Atēla attīrīšana no trokšņiem;](#_qamnlva969pk)
* [Attēla pagriešana.](#_edq14jutbv1g)

Kā arī lietotājam papildus ir iespēja manuāli pielietot šādas metodes:

* [Malu atpazīšana;](#_6w3ypuu96qvu)
* [Segmentācija](#_wlg01djigj0s);
* [Kontrasta pārbaude;](#_lur7hiy0kq4b)
* [Erozija;](#_tn0hy1kvfsxh)

## 3.1 Attēla pārveidošana par melnbaltu

Fotogrāfēti, skenēti vai citādi iegūti attēli, ja vien tie nav uzņemti ar īpašiem iestatījumiem, pārsvarā ir krāsainā formātā. Teksta atpazīšanai attēlā esošās krāsas nav svarīgs faktors, tāpēc, lai atviegloti gan teksta analīzes procesu, gan arī tālākos attēla apstrādes posmus, sistēmā ievadītais attēls tiek pārveidots par melnbaltu. Šādi tiek samazināts kļūdu risks gadījumā, ja vairāko toņu krāsas traucē simbolu atpazīšanas algoritmam.

Papildus tam, melnbaltus attēlus ir vienkāršāk pārveidot un uzlabot to kvalitāti, attīrīt tos no trokšņiem un veikt citas darbības, piemēram, attēla segmentāciju.

Lai attēlu pārveidotu par melnbaltu, sistēma izmanto cv2 bibliotēkas funkcijas:

**cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)**

un

**cv2.bitwise\_not(gray)**

kas pārveido attēla pikseļus uz tiem atbilstošajām pelēkā toņa vērtībām.

## 3.2 Attēla asināšana un Laplasa tests miglainības noteikšanai

Ja uzņemtais attēls ir miglains, tas kļūst grūti analizējams gan sistēmai, gan arī cilvēkam un no šāda attēla nav jēgas. Ja attēls tiek uzņemt ar fotoaparātu vai kādu ierīcē iebūvētu kameru, nav iespējams vienmēr garantēt rezultējošā attēla skaidrību. Kā rezultātā rodas grūti apstrādājami attēli, no kuriem OCR nav spējīgs iegūt informāciju par tekstu.

Miglainību var ietekmēt dažādi faktori kā:

* Kameras izšķirtspēja,
* Kameras lēcas tīrības pakāpe,
* Laikapstākļi attēla uzņemšanas brīdī,
* Fotogrāfa roku trīcēšana,
* kameras izkustēšanās foto uzņemšanas brīdī,
* u.c.

Lai sistēmas OCR programmatūra spētu atpazīt attēlā esošo tekstu, pēc attēla ievadīšanas ir nepieciešams veikt attēla miglainības pakāpes pārbaudi. Tas tiek darīts ar Laplasa miglainības noteikšanas testu.

Laplasa miglainības testam tiek nozīmēts miglainības slieksnis, kas ir konstanta vērtība, pret kuru tiek pielīdzināta attēla vidējā laplasa dispersijas vērtība, kuru iespējams iegūt izmantojot cv2 bibliotēkas metodi (attēli 1., 2. un 3.).

Ja pielīdzinātā vērtība ir lielāka par miglainības slieksni, attēls netiek uzskatīts par miglainu un tas tiek nodots tālākai apstrādei. Taču, ja pielīdzinātā vērtība ir mazāka par norādīto miglainības slieksni, attēls tiek atzīts kā “miglains”.

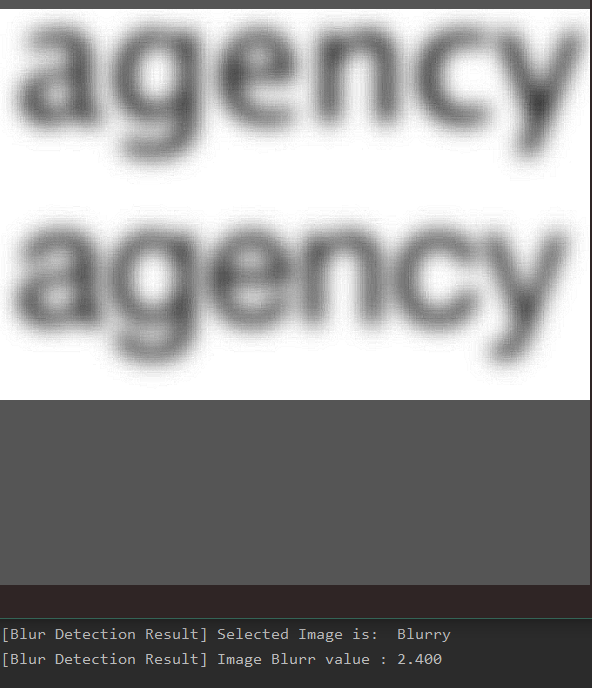


1. attēls: Attēla vidējās laplasa dispersijas vērtības noteikšanas formula no cv bibliotēkas.

Metode pielīdzina katru attēla 3x3 pikseļu matricu sekojošai pikseļu matricai [{0, 1, 0}, {1, -4, 1}, {0, 1, 0}], un no pielīdzināšanas rezultātiem tiek izrēķināta dispersija.



2. attēls: Programmas informatīvā izvade par ievades attēla asumu.



3. attēls: Programmas informatīvā izvade par ievades attēla asumu

## 3.2.1 Attēla asināšana

Attēlu, kurš ir atzīts kā miglains, nepieciešams uzlabot ar attēla asināšanas metodi līdz brīdim kamēr tas vairs nav uzskatāms par miglainu. To sistēmā var panākt pēc asināšanas vēlreiz veicot Laplasa miglainības testa soli. Šādi miglainības pārbaudes un konsekventās attēla asināšanas process atkārtosies līdz attēls vairs nav uzskatāms par miglainu.

Attēla asināšanas metode izmanto kodā definētu masīvu (formula 1)

([[-1, -1, -1],

[-1, 9, -1], (1)

[-1, -1, -1]]),

kā pamata masīvu, kas tiek izmantots kā filtrs. Šis filtrs tiek sistemātiski pielietots katram attēla pikselim, tā rezultāta padarot attēla gaišās daļas gaišākas un tumšās - tumšākas, kopumā palielinot kontrastu starp tām un pārveidojot attēlu skaidrāku.

**3.3 Attēla fona nodzēšana**

Vēl viens no faktoriem, kas var ietekmēt UNTBORAP sistēmā izmantotā OCR simbolu atpazīšanas kvalitāti ir attēla fons. Sistēmai vieglāk ir veikt atpazīšanu attēliem, kuriem fons ir noņemts jeb izdzēsts, jo nepastāv risks mēģināt atpazīt kādu simbolu, kas patiesībā ir tikai fona troksnis vai kāds nenozīmīgs tumšs laukums.

Sistēmas testēšanas gaitā tika secināts, ka, lai sistēmas rezultāti būtu skaidri salasāmi, un, lai netiktu velti tērēta enerģija un resursi izņēmuma gadījumu apstrādei, attēliem būs nepieciešams gaišs fons un tumši burti. Veicot nelielu analīzi ņemot vērā sistēmas mērķi un faktu, ka lietotāji pārsvarā raksta ar tumšas krāsas rakstāmrīku uz gaišas krāsas fona, tika nolemts, ka sistēmas pielietošanā netiks atbalstīti ievadītitie attēli, kuros teksts ir gaišā krāsā, bet fons - tumšā.

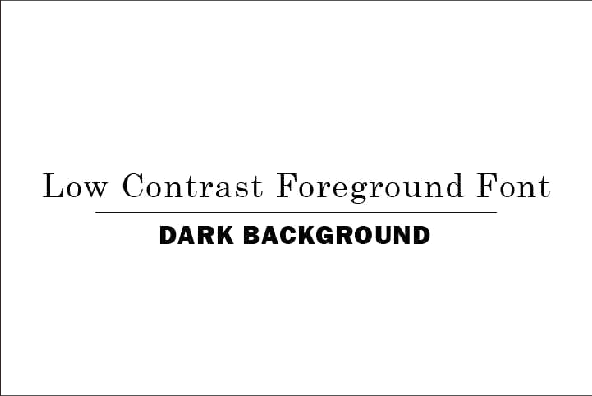
Nodzēšot attēla fonu un atstājot tikai tekstu, sistēmai ir vieglāk atpazīt simbolus, jo netiek ņemti vērā tekstā neiesaistīti krāsas pikseļi, fona dekorācijas un nejauši nofotografēti objekti (piemēram, putekļi vai netīrumi), kas traucētu skaidrai teksta atpazīšanai.

Sistēma attēlam noņem fonu pielietojot automātisko sliekšņošanu ar vērtībām 0 un 255, atbilstoši melnajiem un baltajiem pikseļiem. Sliekšņošanas būtība ir pārvērst jebkuru pikseli tieši melnu vai tieši baltu. T.i. - ja pikseļa krāsas vērtība atrodas sliekšņa pusē, kas ir tuvāks baltajai krāsai, pikseļa krāsas vērtība tiek pārmainīta uz baltu. Piemēri tam, kā sistēmā tiek pielietota sliekšņošana un tās rezultāti redzami 5. un 6. attēlā.

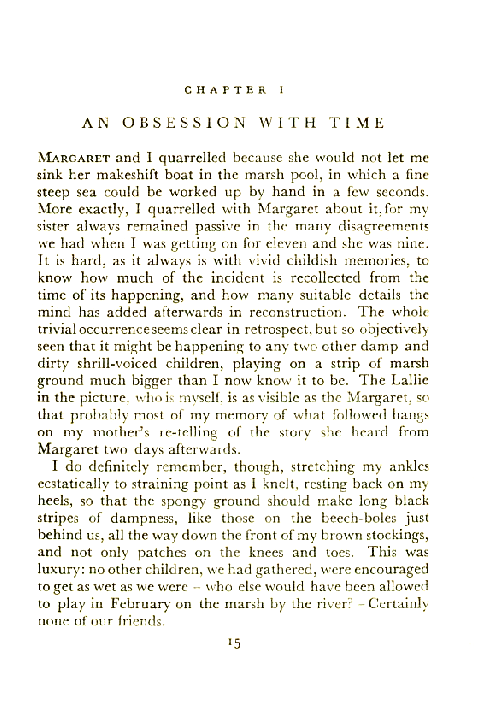
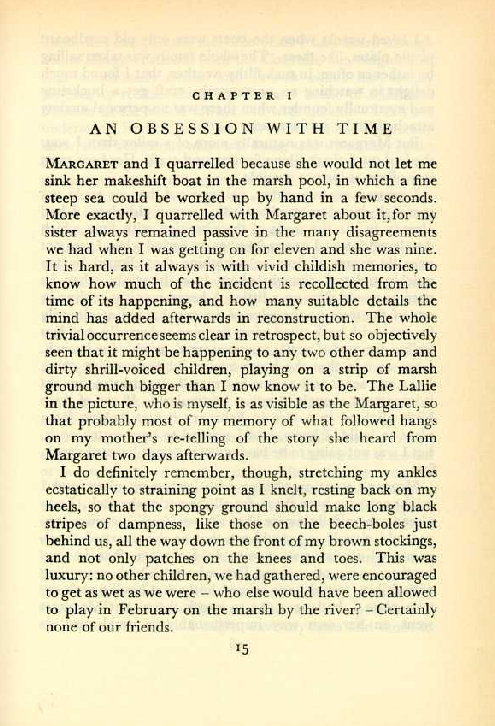
Sliekšņošanai tiek pielietota sekojošā funkcija:

yen\_threshold = threshold\_yen(image)

bright = rescale\_intensity(image, (0, yen\_threshold), (0, 255))



4. attēls: a) Ievades attēls bez fona noņemšanas sliekšņošanas b) Ievades attēls ar fona noņemšanas sliekšņošanu



5. attēls: a) Ievades attēls bez fona noņemšanas sliekšņošanas b) Ievades attēls ar fona noņemšanas sliekšņošanu (*thresholding*)

## 3.4. Attēla kontrasta un spilgtuma korekcija

Viens no visbiežāk pielietotajiem attēlu apstrādes pamata pārveidojumiem ir tā kontrasta un gaišuma mainīšana. Iespēja mainīt uzņemtā attēla gaišumu un kontrastu ir sagaidāma gandrīz no jebkuras ar attēliem saistītas, plaši pazīstamas programmatūras, jo tas ir nozīmīgs veids kā uzlabot vai izmainīt attēla kvalitāti.

Spilgtāks attēls ar augstāku kontrastu ļauj vieglāk noteikt attēlā esošās detaļas un saprast teksta kontrastu. Tas palīdz saprast arī to, kura attēla daļa ir teksts un kura - teksta fons. Gan kontrastu, gan arī spilgtumu ir jāspēj paŗveidot tā, lai rezultējošais attēls vēl aizvien ir lasāms, jo pārāk spilgta vai pārāk kontrastaina attēla konteksts pazūd un attēls tiek piesārņots ar nevajadzīgām vērtībām.

Automātiska kontrasta un spilgtuma korekcija tiek nodrošināta vadoties pēc ievadītā attēla histogrammas. Histogrammai tiek nozīmēti alfa un beta kanāli, kur ɑ apzīmē krāsu vērtību diapozona padziļinājumu un β apzīmē histogrammas diapozona pārvietošanu. Kontrasts un spilgtums tiek automātiski pielāgoti vadotos pēc formulas

O(x,y) = alpha \* I(x,y) + beta.

Rezultējošam diapozonam ir jābūt no 0 līdz 255. Lai veiktu spilgtuma un kontrasta pielāgošanu, tiek “apgriezti” histogrammas sānu posmi, kur esošie krāsas pikseļi sastāda mazāk kā 1% no kopējā attēla, un kur krāsa frekvence ir mazāka par sliekšņa vērtību, kas tiek noteikts balstoties uz iepriekš norādītām min un max vērtībām. Pēc šīs darbības tiek aprēķināti augstāk minētie ɑ, kas nosaka kontrastu un β, kas nosaka spilgtumu, kanāli, pēc formulām:

ɑ= 255 / melnbaltā attēla histogrammas max vērtība - melnbaltā attēla histogrammas min vērtība

β= - melnbaltā attēla histogrammas min vērtība \* ɑ

Piemēru tam, kā attēlu ietekmē automātiskā gaišuma un kontrasta maiņa, var redzēt 6. attēlā.



6. attēls: a) Ievades attēls bez attēla kontrasta un spilgtuma korekcijas b) Ievades attēls ar attēla automātisko kontrasta un spilgtuma korekciju

## 3.5. Attēla attīrīšana no trokšņiem

Digitāli uzņemtiem, skenētiem vai apstrādātiem attēliem vienmēr ir risks zaudēt kvalitāti dažādu ārējo ietekmju dēļ, un trokšņu esamība attēlos būtiski samazina iespēju atpazīt attēla saturu, izmantojot datorredzi. Trokšņa tipi ir daudz un dažādi, bet šajā darbā sīkāk tika apskatīti tikai divi pazīstamākie trokšņi: Sāls un piparu troksnis, un Gausa troksnis.



7. attēls: a) Ievades attēls bez attēla attīrīšanas no trokšņiem b) Ievades attēls ar attēla automātisko attīrīšanu no trokšņiem

**3.5.1 Sāls un piparu troksnis**

Sāls un piparu troksnis izpaužas kā sabojāti balti pikseļi apgabalos, kur attēls ir tumšs, un melni pikseļi, kur attēls ir gaišs. Attēls pilnībā netiek sabojāts, taču tā kvalitāte samazinās. Šādus attēlus visbiežāk labo pārbaudot katru attēla pikseli.

Ja tiek atrasts pikselis ar vērtību 0 vai 255, no šī pikseļa, kā centra, tiek atlasīts neliels attēla apgabals (parasti tas ir 3x3 pikseļi liels). Atlasītajā apgabalā tiek pārbaudīts, vai šim pikselim apkārt esošie pikseļi ir tādā pašā krāsu vērtību diapazonā. Ja apkārt esošie pikseļi visi atrodas gaišas krāsas vērtības diapozonā, bet centra pikselis ir melns, tas visticamāk ir sabojāts pikselis, kas radies trokšņa rezultātā. Šī pikseļa vērtību nomaina uz vērtību, kas ir atbilst pikseļa blakus esošo pikseļu vērtību mediānai. Ja apgabalā tiek atrasts vēl kāds trokšņa pikselis, tad centra pikseļa vērtību nomaina uz to apgabala pikseļu vērtību mediānu, kuri nav trokšņaini.

Šādi tiek analizēts viss attēls, līdz trokšņaino pikseļu vairs nav.

**3.5.2 Gausa troksnis**

Pazīstams arī kā dabiskais troksnis, Gausa troksnis rodas dabisku parādību rezultātā, piemēram, temperatūras ietekmē vibrējošu atomu un neliela siltumstarojuma no siltiem objektiem ietekmes rezultātā. Gausa troksnis pārsvarā izmaina digitālā attēla pelēko pikseļu vērtības, kā rezultātā Gausa trokšņa modeli nosaka tā PDF (Probability Density Function, jeb Varbūtības ciešuma funkcija), vai tas tiek normalizēts kā histogramma, ar uzsvaru uz pelēkajām vērtībām.

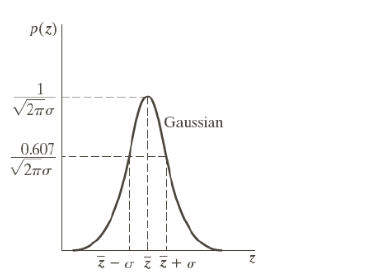
Gausa trokšņa modeli iespējams aprakstīt kā formulu:

,kur z norāda trokšņa intensitāti,

norāda z vidējo aritmētisko,

σ norāda trokšņa standarta novirzi, kas, kāpināta otrajā pakāpē (), tiek saukta par variāciju.

Gausa trokšņa modelis tiek izmantots lai atspoguļotu vispārīgus reālus scenārijus. Šajā modelī intensitātes vidējā aritmētiskā vērtība ir nulle; variācija ir 0,1 un PDF ir 256 pelēkā līmeņi. Šo modeli var redzēt 8. attēlā.



8. attēls: Gausa trokšņa PDF modelis.

**3.5.3 Gausa trokšņa noteikšanas algoritms**

Pieņem, ka trokšņainais attēls tiek definēts kā A. Pārbaudāmais pikselis, jeb testa pikselis tiek uzdots ar koordinātām (i, j). Par pārbaudāmo apgabalu, līdzīgi kā Sāls un piparu troksņa noteikšanas metodē, var ņemt 3x3 pikseļu lielu laukumu, bet šajā darbā tiek apskatīts 5x5 pikseļi liels laukums, kura centrā, protams, atrodas testa pikselis.

5x5 pikseļi lielais laukums tiek sadalīts mazākos, 3x3 pikseļus lielos apakšlaukumos tā, lai testa pikselis tiktu iekļauts katrā no apakšlaukumiem.. Katram apakšlaukumam tiek izrēķināta standarta novirze σ. Par standarta novirzes atsauci tiek uzskatīta apakšlaukumu standarta novirzes mediāna no vērtības (σi), i=1, 2 … 9, (9 = N = apakšlaukumu skaits), tiek noteikts novirzes minimums σmin un maksimums σmax un tiek izrēķināts deviņu apakšlaukumu standarta novirzes vidējais aritmētiskais (σavg), kuru novirze pieder kopai [σmin, σmax].

Šo vidējo aritmētisko standarta novirzes vērtību izmanto, lai noteiktu vai testa pikselis ir trokšņains. Pārbaude tiek veikta salīdzinot testa pikseļa (i, j) vērtību ar tam apkārt esošā 3x3 pikseļu lielā apakšlaukuma vidējo aritmētisko vērtību. Pikselis uzskatāms par trokšņainu, ja tā atšķirība no vidējā aritmētiskā pieder kopai [a,b], kur diapozona robežas ‘a’ un ‘b’ tiek iegūtas eksperimentāli, kā 0.5\*σavg un 0.5. Šo procesu atkārto viiem trokšņainā attēla pikseļiem.

Ja testa pikselis tiek noteikts kā trokšņains, tas tiek aizstāts ar jaunu vērtību, kuru iegūst no formulas:

xjaunais(i,j) = [- 0,5σavg], kur

xjaunais ir pikseļa jaunā vērtība (i,j),

ir centrālā 3x3 pikseļu lielā apakšlaukuma vidējais aritmētiskais,

σavg ir standarta novirzes vidējais aritmētiskais.

**1.9. Attēla trokšņa līmeņa noteikšana**

Lai gan ar cilvēka redzi un saprašanu ir salīdzinoši viegli noteikt, kuri attēli ir trokšņaini un kuri nav, sistēma to nevar noteikt tik vienkārši. Nav lietderīgi uzreiz no trokšņiem attīrīt visus attēlus, kas tiek ievadīti programmā, tapēc no sākuma ir nepieciešams noteikt vai ievadītais attēls vispār ir uzskatāms par trokšņainu.

Viens no trokšņa līmeņa noteikšanas veidiem ir saskaitīt cik no attēla pikseļiem ir trokšņaini un izmantot šo vērtību procentuāli pret attēla kopējo pikseļu skaitu.

Melnbaltiem attēliem ir metode, kas salīdzina no formulas (formula 2) iegūtu sigma vērtību ar noteiktu robežu.

 (2)

**1.10. Trokšņa samazināšana**

Attēls jau ir melnbaltā formā, tapēc tā trokšņa samazināšanai tiek pileietota cv2 bibliotēkas metode fastNlMeansDenoising(). Metode sistemātiski pārbauda katru attēla pikseli, atlasot nelielu pikseļu matricu apkārt izvēlētajam pikselim, meklē attēlā šai matricai līdzīgus apgabalus, iegūst vidējo vērtību no visām matricām un aizvietu pārbaudāmo pikseli ar šo iegūto vērtību. Šo metodi sauc arī par Ne Lokālo vidējo Attrokšņošanu (Non-Local Means Denoising).

Jāpiemin, ka, par cik algoritms nespēj saprast atšķirību starp troksni un smalkām attēla detaļām, pārāk liela trokšņa samazināšana var no attēla attīrīt šīs detaļas, kā mazas līnijas, tekstūras vai punktus, kas savukārt var izmainīt attēla kontekstu.

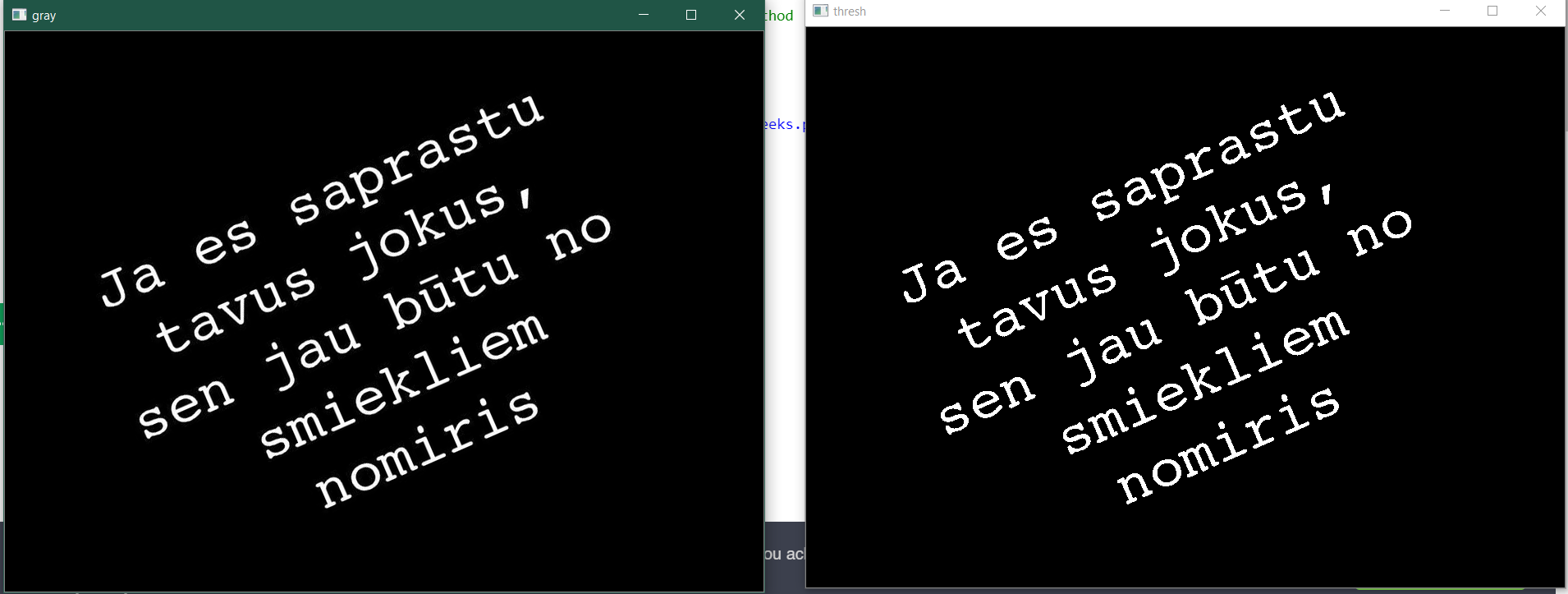
## 3.6 Attēlā esošā teksta pagriešana

Lai gan pārsvarā var pieņemt, ka lietotājs, fotografējot vai skenējot tekstu priekš UTBORAP sistēmas, centīsies iegūt attēlu, kur teksts ir horizontālā stāvoklī, nedrīkst ignorēt iespēju, kad ievadītajā attēlā teksts var būt šķībs vai vertikāls. Šādos gadījumos sistēmai ir jāspēj atpazīt tekstu kā šķību, un jākoriģē tas tā, lai attēlā teksts būtu horizontālā stāvoklī, ļauot OCR funkcijām darboties netraucēti.

Tas tiek panākts vispirms melnbaltam attēlam samainot krāsu vērtības uz pretējām, tā panākot ka attēla fons tagad ir melns, un pats teksts - balts. Lai iegūtu vēl skaidrāku attēlu, tam tiek pielieto sliekšņošanu, kas visiem fona pikseļiem piešķir vērtību 0, un visiem teksta pikseļiem - vērtību 255 (skat 9. Un 10. att.). Tad tiek iegūtas visu to pikseļu koordinātas, kuru krāsas vērtība ir lielāka par 0, un tiek konstruēts pagriezts taisnstūris tā, lai tajā tiktu iekļauti visi pikseļi.

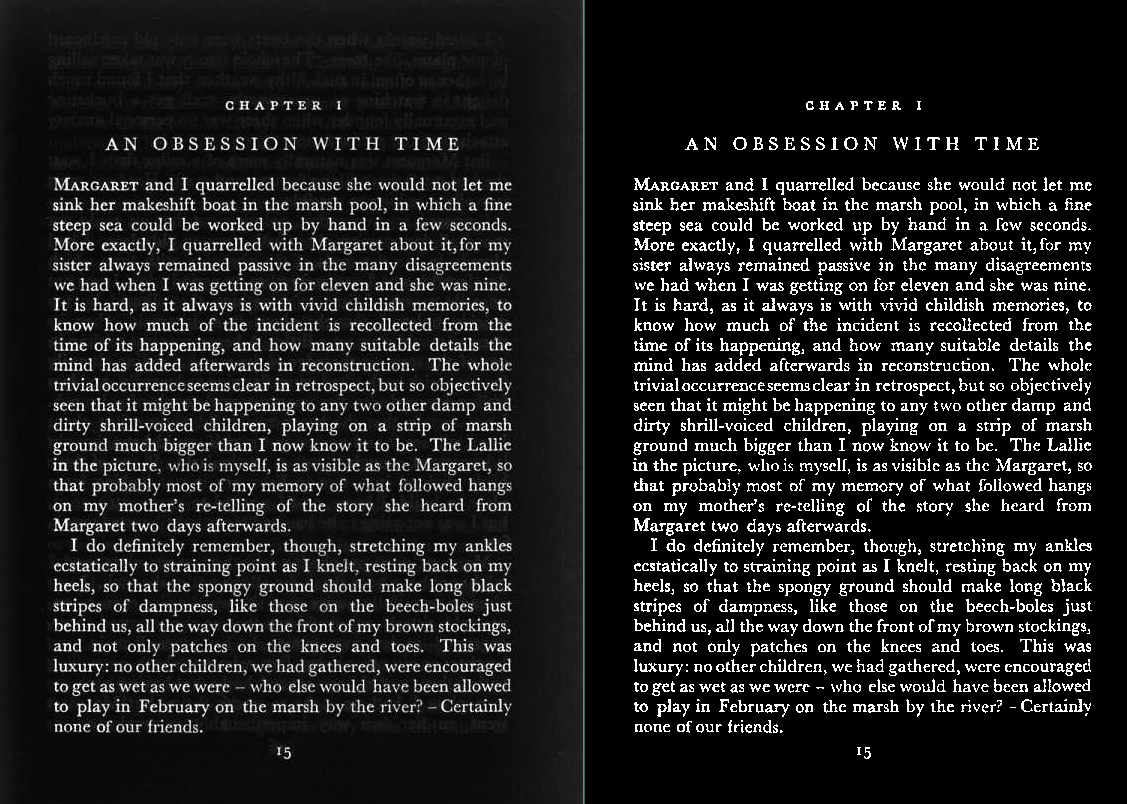
Pēc tam konstruē vēl vienu identisku taisnstūri, kas atrodas horizontālā pozīcijā. Tiek iegūtas koordinātas abiem taisnstūriem un, gadījumā, ja tās atšķiras, pirmais taisnstūris kopā ar tajā esošo tekstu, tiek pagriezti lai atbilstu otrajam taisnstūrim.

Ja tiek izmantots attēls ar caurspīdīgu fonu, rezultējošais attēls var būt ne pilnīgi taisnā leņķī, vai ar nevēlamām modifikācijām (skat 13-14 att.), tāpēc ieteicams izmantot attēlus ar necaurspīdīgiem foniem. Pietam, ja caurspīdīgu fonu saturošs attēls oriģināli ir vertikālā pozīcijā, pēc pagriešanas metodes izpildes, tas vēl aizvien būs vertikāls.





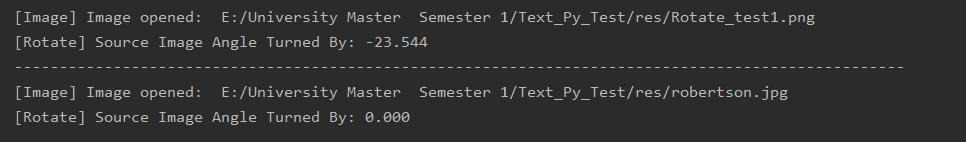
9. attēls: a) Gray izvade b) sliekšņošanas izvade c) Pagriezts teksts



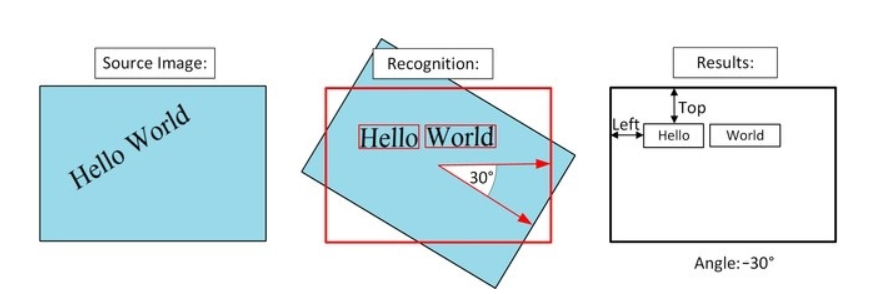
10. attēls: a) Gray izvade b) Sliekšņoņas izvade

Rezultātu vērtību izvade tiek veikta izmantojot koda rindu:

**print("[Rotate] Source Image Angle Turned By: {:.3f}".format(angle)) ← lenķa izvade**

****

**11. attēls:** Attēla pagriešanas lenķa informatīvā izvade

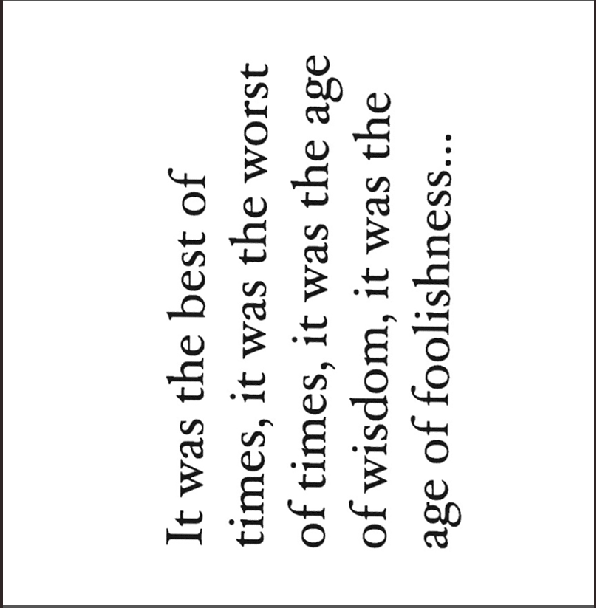
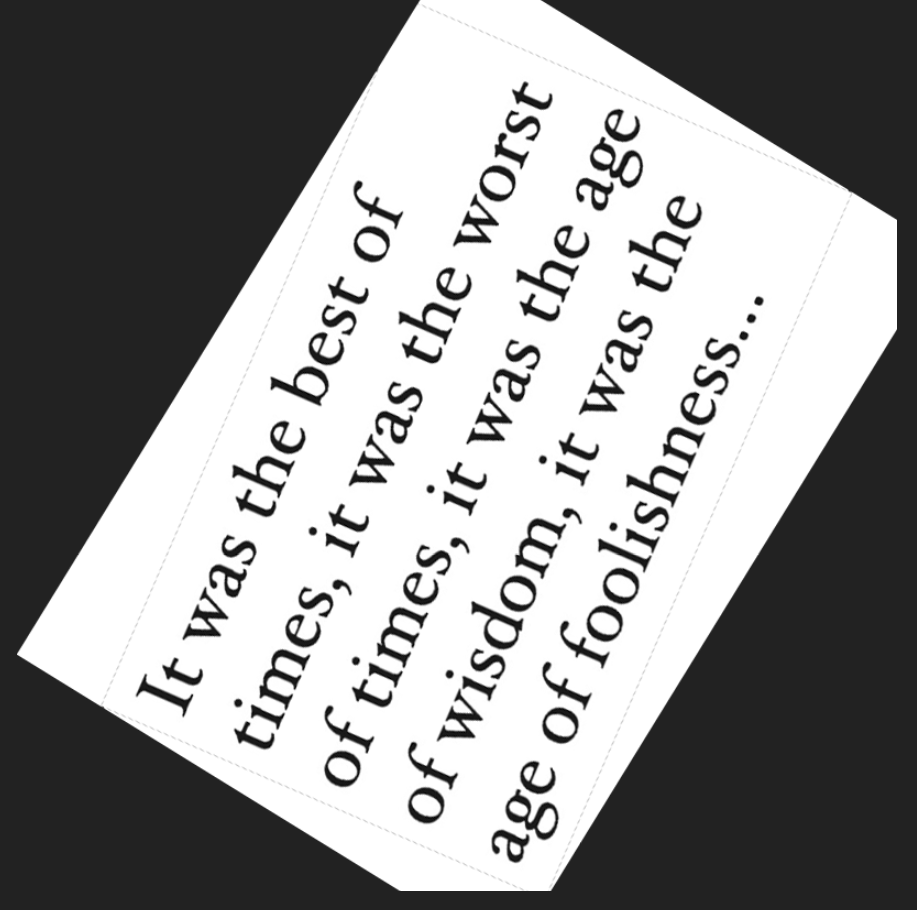


12. attēls: Gausa trokšņa PDF modelis. [**Avots:** <https://codeburst.io/optical-character-recognition-recognizing-text-to-labels-on-an-android-platform-4c20bddc9175>]

Pagriešanas metode nestrādā, ja ir transparent background! (vai arī strādā, bet ar nepilnībām). Rezultāts var būt **ne līdz galam pagriezts.**

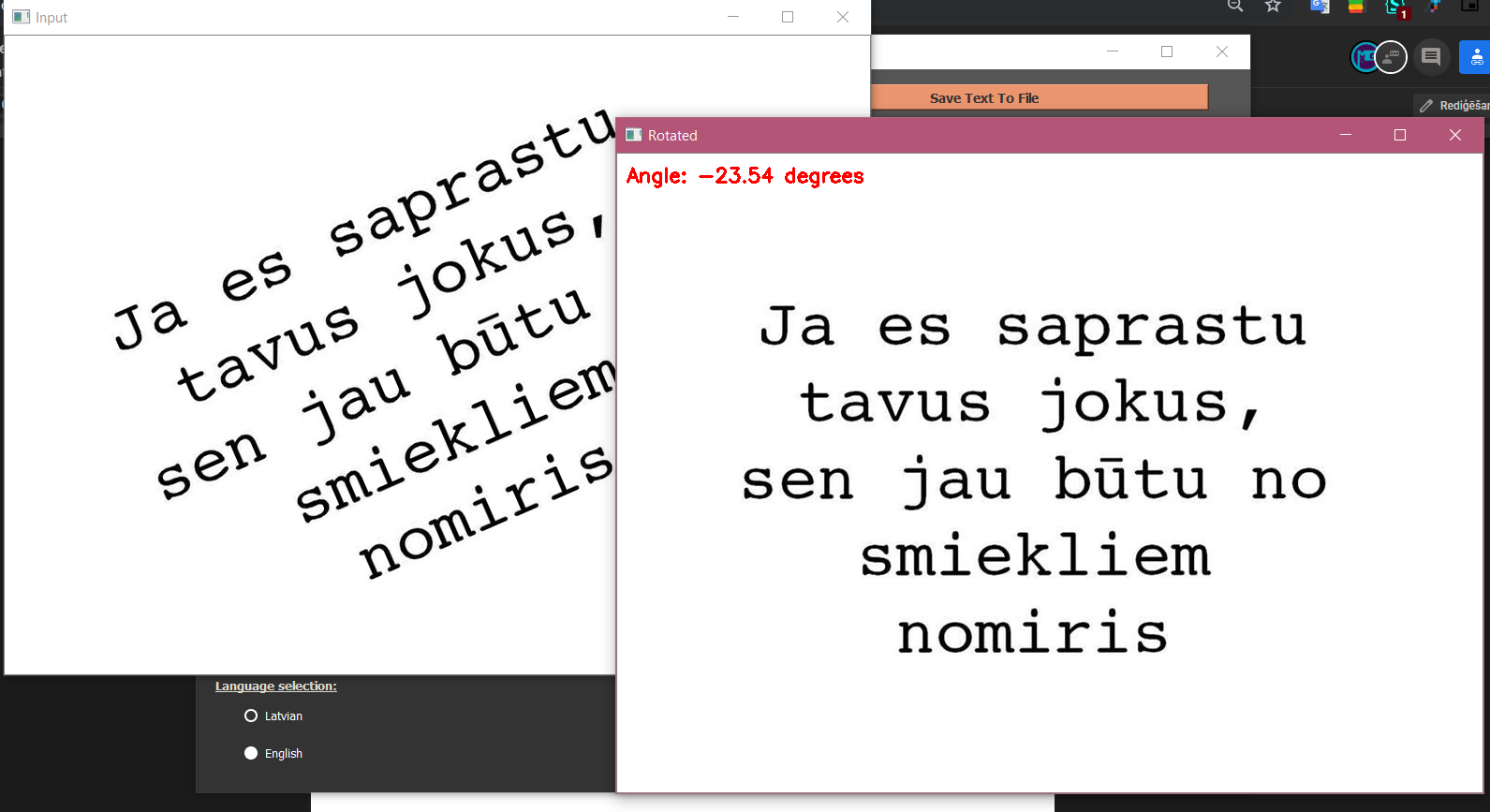


13. attēls: a) ievades attēls b) pagrieztais attēls

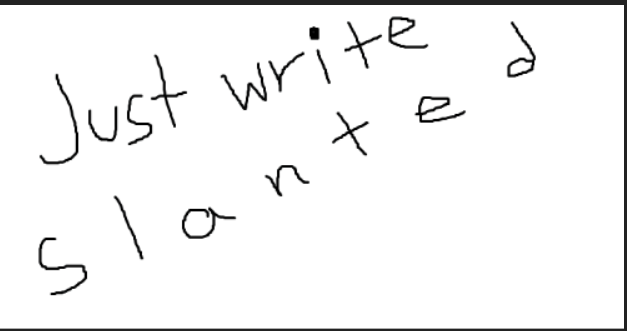


14. attēls: a) ievades attēls b) pagrieztais attēls

Zemāk ir redzami vairāki attēla pagriešanas rezultātu piemēri.



15. attēls: a) Ievades attēls bez automātiskās attēla pagriešanas b) Ievades attēls ar automātisko attēla pagriešanu

****

16. attēls: a) Ievades attēls bez automātiskās attēla pagriešanas b) Ievades attēls ar automātisko attēla pagriešanu

# 4. Neironu tīklu apmācība burtu atpazīšanai

## 4.1. OCR Stratēģija

Teksta atpazīšana ir 2 soļu uzdevums. Pirmkārt ir nepieciešams atpazīt tekstu attēlā, otrkārt - identificēt simbolus izmantojot 3 pamatpieejas:

1. Datorredzes metodes
2. Standarta dziļās apmācības (deep learning) pieeja
3. Padziļināta dziļā apmācība

[]<https://missinglink.ai/guides/tensorflow/building-tensorflow-ocr-systems-key-approaches-and-tutorials/>

Tesseract un visas tā trenēšanai nepieciešamās bibliotēkas tika instalētas izmantojot virtuālo mašīnu, kuras operātājsistēma ir Ubuntu 18.04. un kuras pamata atmiņa ir 2048 MB.

Programmatūra un bibliotēkas, kas nepieciešamas Tesseract neironu tīkla apmācībai un lietojumprogrammatūras izstrādei:

* Oracle VM VirtualBox Manager
* Ubuntu 18.04
* Tesseract OCR
* Leptonica
* OCR-D
* Python
* OpenCV
* Git (priekš versiju kontroles un komandas darba)
* Valodu datu faili

Neironu tīkli plaši tiek izmantoti numurzīmju, tekstu, zīmju, objektu atpazīšanā kā arī dažādu mērķu attēlu apstrādes processos, piemēram segmentācijā, malu noteikšanā, vai arī lai pārklātu papildus informāciju pāri atpazītajam objektam vai datiem, automatizētu kādu processu, kā, piemēram aprēķinot atpazīto nummuru summu.

**Pamatsoļi neironu tīkla izveidei un apmācībai:**

1.Apmācības failu izstrāde un sagatavošana

2. Informācijas nolasīšanas skripta izveide

3. Apmācības tīkla izstrāde (skriptu izveide)

4.Neironu tīkla apmācība

5.Tīkla papildus apmācība un precizitātes uzlabošana

6.Apmācītā neironu tīkla darbības testēšana

## 4.2 Apmācības datu sagatave

Uz latīņu valodas bāzētām valodām esošais modelis jau ir apmācīts ar apmēram 400000 teksta rindiņām, kas satur apmēram 4500 dažādus fontus. Priekš citiem valodu skriptiem tik liela fontu pārstāvniecība nav pieejama, taču arī šīs valodas ir ietrenētas ar līdziga daudzuma teksta rindiņām.

Augstas precizitātes ar roku rakstīta teksta apmācībai nepieciešams ļoti liels skaits ar apmācības datiem (aptuveni 100 000 attēlu).

Tesseract iespējamie ievades failu formāti:

* TIFF (vēlamais)
* JPG
* PNG

Tesseract izvades failu formāti:

* Plain text (utf-8)
* PDF
* HTML
* hOCR

Pirmais treniņa posms ir pietiekami liela apjoma datu sagatavošana treniņam. Tāpat kā ar pamata Tesseract, ir iespēja vai nu renderēt sintētiskus treniņa datus no fontiem, vai izmantot nosaukumu piešķiršanu iepriekš eksistējošiem attēliem. Jebkurā gadījumā, tesseract treniņa datiem nepieciešamais formāts ir tiff/box faila pāris, izņemot to, ka box failiem pietiek at teksta rindiņu un nav nepieciešami atsevišķi simboli.

Treniņa datu faili tika sadalīti attēlos (“\*.tif”) un attēla teksts tika sagatavots atsevišķi un saglabāts kā “\*.gt.txt” teksta fails ar attēlam atbilstošo faila nosaukumu. Failā esošajam tekstam ir jābūt vienā rindiņā un vienkāršam, un faila nosaukumam ir jābūt tādam pašan kā rindiņas attēlam, tikai faila formāts jānomaina no .tif uz .gt.txt.

Lai renderētu attēla box failu, izpildāmajā skriptā tika iekļauta sekojošā koda rinda:

* tesseract [image name including extension] [box file name] lstmbox

Nākošais solis ir Unichar faila izveidošana LSTM faili ir lieliski piemēroti mācīšanās cikliem, taču tie paliek aizvien lēnāki palielinoties dažādo statusu skaitam. Ir pieejami empīriski rezultāti, kas norāda, ka ir efektīvāk likt LSTM iemācīties garu darbību secību, nekā mācīt tam īsu darbību secību caur vairākām klasēm. Tāpēc sarežģītiem skriptiem, tā vietā lai izmantotu lielu daudzumu klašu, ir ieteicamāk pārkodēt katru simbolu kā īsu kodu secību ar nelielu skaitu klašu.

Kad no unichartset un iespējamiem direktorijas datiem tiek izveidoti sākuma treniņdati, ir nepieciešams nodefinēt ceļu uz sākumdatu failu, kas satur unichartset, ierakstītāju un ieteicamo valodas modeli.

Ir pieejamas vairākas metodes kā veikt tīkla trenēšanu. Palaiž Tesseract, lai tas apstrādā attēlu un tā box failu, lai izveidotu treniņa datu komplektu, pielietojot Tesseract treniņa bibliotēkas. Treneris periodiski saglabā marķierus, izmantojot **--model\_output** kā pamata nosaukumu. Līdz ar to ir iespējams apstādināt trenēšanas procesu jebkurā brītī, un atsākt to vai turpināt procesu ar pievienotiem treniņa datiem, izmantojot to pašu komandas rindiņu. Lai uzspiestu treniņa restartēšanu, izmanto citu **--model\_output** vai izdzēš visus failus.

Treniņa datus ieteicams testēt uz atsevišķas treniņu datu kopas. Ja rezultējošie dati nav tik precīzi cik tas nepieciešams, jāveic papildus treniņi, līdz ir sasniegti vēlamie rezultāti.

## 4.3. Tesseract apmācība izmantojot sagatavotos datus

Ātrākais veids kā izveidot treniņa datus, izmantojot jaunu kopu treniņdatiem paredzētu attēlu, ir izmantojot OCR-D un ievietojot NAME.tif un NAME.gt.txt failus iekš pamata izmantotās mapes.

* git clone <https://github.com/OCR-D/ocrd-train.git>

Priekš Windows operētājsistēmas, nepieciešams pārliecināties, la “tesseract-OCR.exe” fails un valodas datu faili visi ir iekš OCR-D mapes, jo savādāk šī metode nedarbosies.

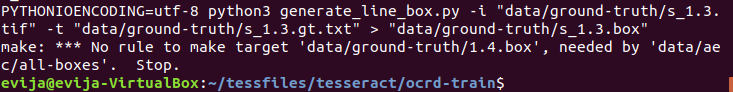
* cd /tessfiles/tesseract/ocrd-train
* export TESSDATA\_PREFIX=~/tessfiles/tesseract/tessdata
* tesseract --list-langs

Ir svarīgi piezīmēt, la Tesseract iespējams pavaicās exportēt TESSDATA\_PREFIX failus katru reizi lad tiek atvērts komand konsoles logs. Ja tesseract atpazīs vismaz vienu valodu iekš tā tessdata failu mapes, treniņš varēs tikt turpināts. Lai uzstāktu modeļa treniņu, nepieciešams izpantit komandu:

* make training

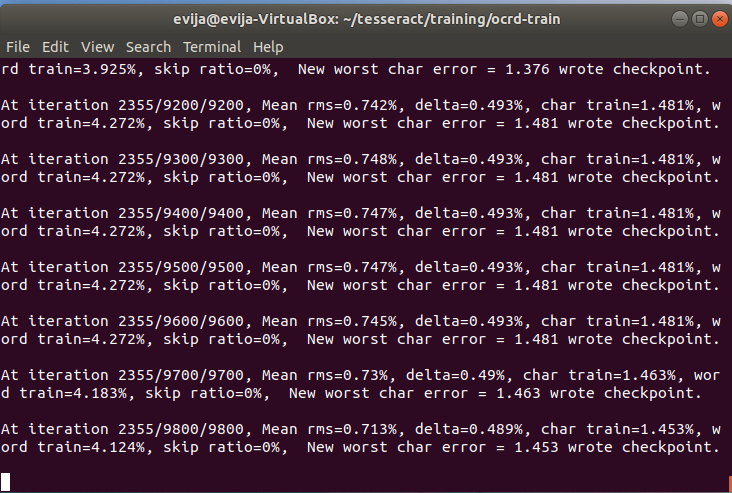
Ja brīdī, kamēr tiek veikta datu trenēšana, tiek uzrādīta kļūda UnicodeEncodeError, nepieciešams pāriet no python šifrētāja uz utf8 un kļūda pazudīs. Pārmaiņu var veikt izmantojot kodu export PYTHONIOENCODING=utf8.

Ja kādam no treniņa attēliem nav atbilstošā “.gt.txt” faila, tiks uzrādīts sekojošais ziņojums, un treniņa process tiks apstādināts:

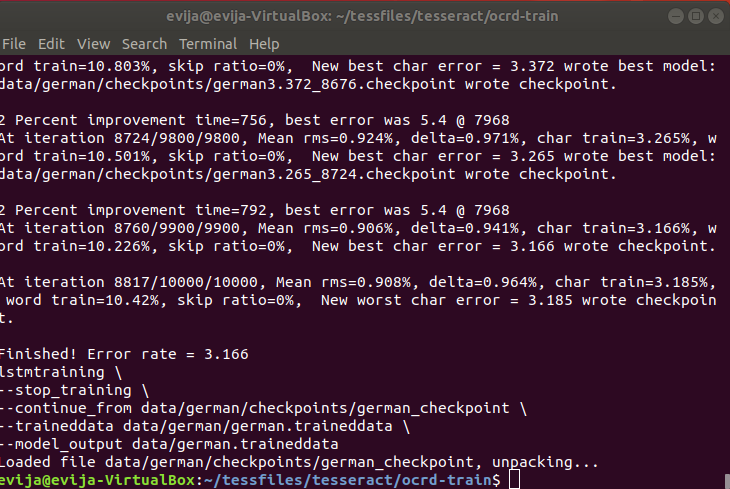


17. attēls: Kļūdas ziņojums, ja netiek atrasts teksta fails ar tādu pašu nosaukumu kāds ir attēlam.

Treniņa process ar tam doto Ģermāņu simbolu datubāzi aizņem līdz pat divām stundām, atkarībā no tā kāds ir maksimālais iterāciju skaits, cik modelim ir noteikts iziet treniņa procesā. Šī treniņa laikā, ,pdelim tika uzdots veikt 10000 iterācijas.

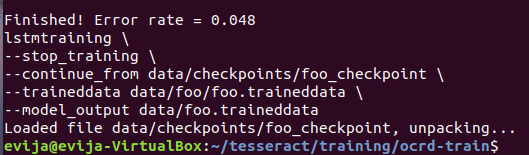


18. attēls: Tesseract datu apmācības processa logs



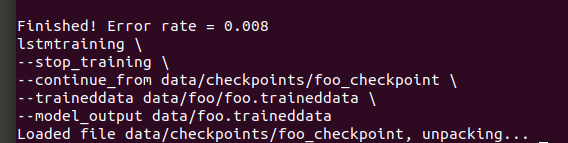
19. attēls: Tessaract datu apmācības procesa rezultāti

Pirmā treniņa procesa laikā, modeļa beigu kļūdu skaits bija 0,048%.



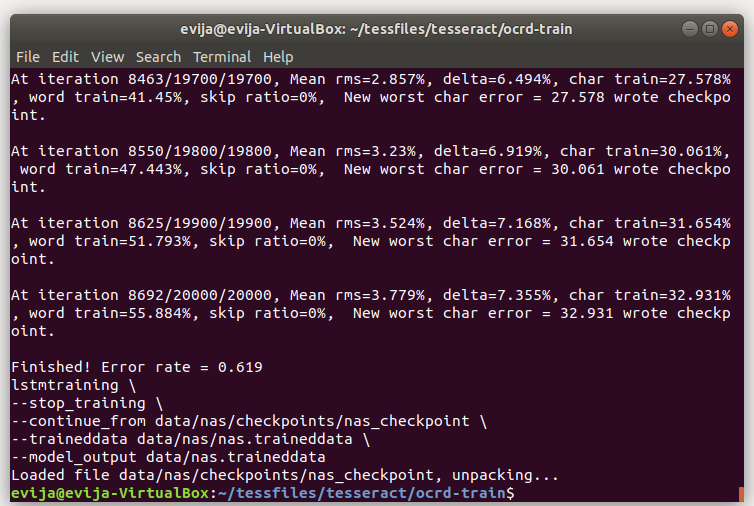
20. attēls: Tesseract apmācības processa izpildes beigas termināla logā

Izmantojot šo modeli un to apmācot atkārtoti ar izveidotajiem rokraksta attēliem tika iegūti precīzāki rezultāti ar 0.008 % beigu apmācības kļūdu intervālu.



21. attēls: Tesseract apmācības processa izpildes beigas termināla logā

Papildus tam, lai pārbaudītu Tesseract apmacības iespējas, tika izveidots komplekts ar 200 attēliem, kas saturēja ar roku rakstītu tekstu, un tiak veikta apmācība ar tiem. Attēli tika trenēti 20000 iterācijās. Šī treniņa rezultāti ir sekojošie:



22. attēls: Tesseract apmācības processa izpildes beigas termināla logā

Lai gan uzrādīto kļūdu gadījumi ir mazāk kā 1%, apmācītā modeļa trenēšana ar jaunajiem, ar roku rakstītā teksta attēliem, nebija tik veiksmīga.. Modelis tā rezultātā spēja atpazīt atsevišķus simbolus, taču nespēja uztvert vārdus, jo, lai apmācīt Tesseract atpazīt konkrētu rokrakstu, nopietnai apmācība būtu nepieciešams milzīgi liels daudzums treniņa datu.

**Tesseract rezultātu optimizēšana**

Ir pieejami vairāki veidi kā tālāk optimizēt gan neironu tīkla apmācībai izmantotos treniņa datus, gan apmācītā datu modeļa precizitāti.

* **Izmēra optimizācija:** Balstoties uz oficiāliem avotiem, lai Tesseract veiksmīgi apstrādātu attēlu, tā optimālais pikseļa izmērs ir 300 DPI. Lai panāktu šo DPI, nepieciešams apstrādāt attēlu izmantojot -r parametru. Palielinot DPI palielinās arī attēla apstrādes laiku.
* **Lapas rotācija:** Ja brīdī, kad tika skenēta lapa, tā nebija optimālajā rotācijas stāvoklī un pēc apstrādes tā paliek 180° vai 45° rotācijas leņķī, Tesseract precizitāte samazināsies.
* **Malu noņemšana:** Atsaucoties uz Tesseract oficiālo lietotāja rokasgrāmatu, malas var tiks kļūdaini uzskatītas par simboliem, it īpaši ja tās ir tumšā krāsā, ar gradiācijas variācijām. Noņemot attēla malas iespējams panākt augstāku Tesseract precizitāti.
* **Trokšņa attīrīšana:** Balstoties uz Tesseract datiem, troksnis ir “neparedzēta krāsu vai silgtuma variācija attēlā”. To iespējams noņemt izmantojot binarizācijas soli, kas polarizē attēla krāsas.

# 

# 5. Lietotnes GUI izveide

## 5.1 QT designer programmatūra

*Qt Designer* ir Qt līdzeklis, kas paredzēts grafisko lietotāja saskarņu (GUI) veidošanai un dizainēšanai izmantojot [Qt logrīkus (Widgets](https://doc.qt.io/qt-5/qtwidgets-index.html)). Ir iespējams komponēt un pielāgot logus un dialoglodziņus pēc kā-redzi-tā-arī-būs (what-you-see-is-what-you-get (WYSIWYG)) pieejas, un tos testēt, izmantojot dažādus ekrānu stilus un izšķirtspējas.

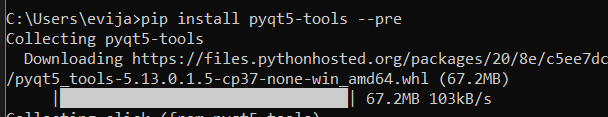
Logrīki un formas, kas tiek weidotas ar *Qt Designer* bez problēmām integrējas iekš programmētā koda, izmantojot Ot signālu un slotu mehānismu, nodrošinot lietotājam vieglu veidu kā noteikt interfeisā esošo elementu uzvedības. Visas *Qt Designer* norādītās īpašības var tikt dinamiski mainītas iekš koda. Un papildus tam iespējas kā logrīku paaugstināšana un pielāgoti spraušņi ļauj lietotājam izmantot savas komponentes kopā ar *Qt Designer*.

## 5.2 Step by step installation and design process

1.**Jāinstalē Python** (Darba laikā tika izmantota jaunākā Python 3.7 versija. )

**2.Instalē Python GUI izstrādes rīku PyQt5, izmantojot komandlīniju konsoli (cmd):**

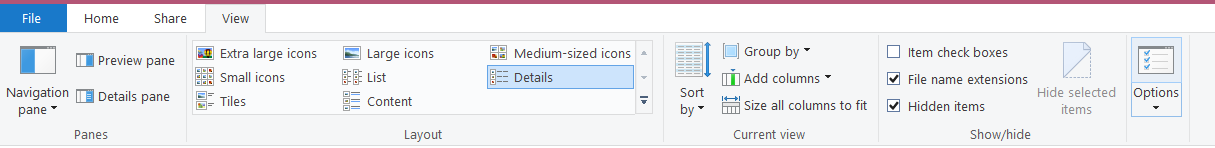
*pip install pyqt5-tools --pre*



23. attēls: Python instalēšana ar cmd

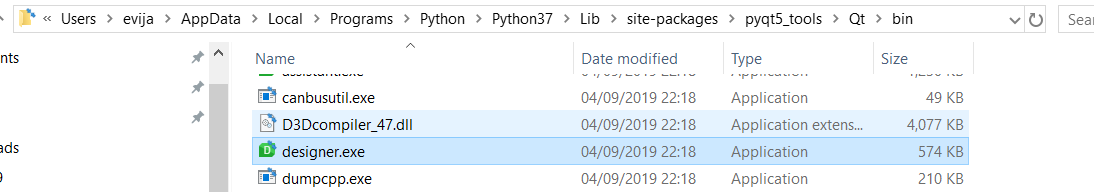
**3. Atrod mapi, kurā tas tika instalēts un atver tajā esošo designer.exe :**

Projekta izstrādei tā bija C:\Users\**USERNAME**\AppData mape, kas pēc noklusējuma ir paslēpta drošības iemeslu dēļ, tapēc to ir nepieciešams iespējot, lai tā kļūtu redzama:



24. attēls: Slēpto failu konfigurēšanas atzīme

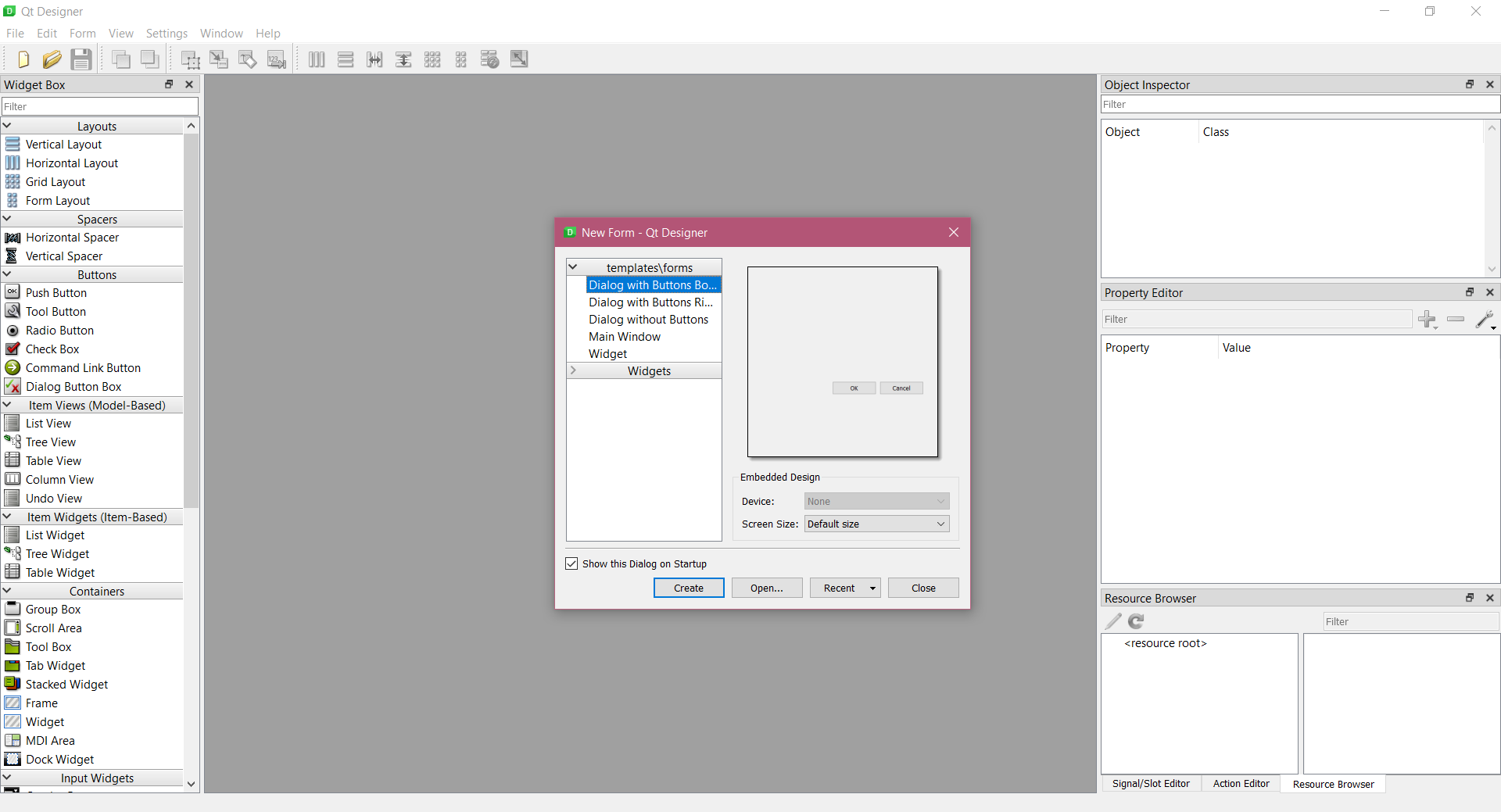
Vai iespējams mapi padarīt veicot atzīmi lauciņā (''Hidden items”):



25. attēls: Designer.exe programma

Balstoties uz python versiju un instalācijas mapi, ceļam būtu jābūt līdzīgam ar šo:. Designer.exe is found in the Lib\site-packages\pyqt5\_tools\Qt\bin path (Vieglai piekļūšanai ir ieteicams izveidot šī ceļa īsceļu uz darbavisrmas.):

4. **Dizainēšanas sākšana :**



26. attēls: Dizainēšanas sākšana

## 5.3 Python kodu izveide un dizaina integrācija

Pēc UI veiksmīgas izveides, ir nepieciešams konvertēt izveidoto “\*.ui” failu par “\*.py” failu. Vispirms tiek izveidota mape, kur tiks glabāts projekts. Tad tajā tiek ievietots “\*ui” fails un caur cmd direktorija tiek izmainīta uz noteikto mapi.

Lai konvertētu failu izmanto .exe file: XXX

Nokopē pyuic5.exe failu iekš Tesseract atpazīšanas direktorijas (Faila nosaukumu var brīvi mainīt) un palaiž šo kodu

(noderīgs video:<https://www.youtube.com/watch?v=c2k1eVjFOw8> )

Var izveidot .bat failu ar šo komandu lai nebūtu tā pati komanda jāraksta vairākkārt, kamēr tiek implementētas dizaina izmaiņas.

Konvertētais python skripts atradīsies direktorijā.

Tālākai izstrādei ir ieteicams izmantot PyCharm - bezmaksas programmatūra prieks izstrādes Python valodā, kas darbojas līdzīgi Visual Studio.

[1] PyCharm lejuplādēt iespējams: <https://www.jetbrains.com/pycharm/>

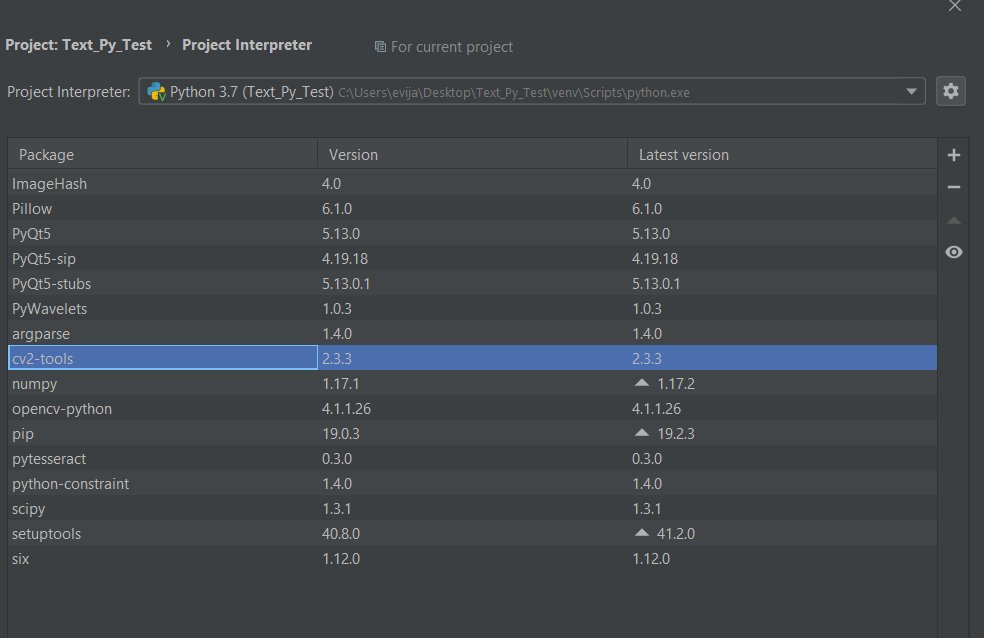
Tagad var sākt rakstīt topošās lietotnes kodu. Vispirms tiek izveidota mape, kurā tiks glabāti visi tālākai izstrādei nepieciešamei faili. Iekš šīs mapes tiek ieliks izveidotais UI fails, konvertētais PY fails. Priekš automatizēšanas tiek pievienots konvertēšanas .bat fails un pyuic5.exe fails. Šie faili palīdzēs implementēt jebkuras izmaiņas ātri un efektīvi, un ļaus mainīt UI reizē ar tā funkcionalitātes iestrdāšanu.

Tātad, pilnais process ietver: PyCharm un QtDesigner.

Soli pa solim integrācijas izpilde izskatās šādi:

1. Saglabā \*.ui failu.
2. Aktivizē \*.bat failu, konvertējot izmaiņas uz \*.py failu.
3. Izmaiņas tiek iestrādātas un ielādētas iekš PyCharm projekta, ļaujot tālāku izstrādi.

## 5.4 Nepieciešamās bibliotēkas



27. attēls: Nepieciešamo bibliotēku saraksts

# 

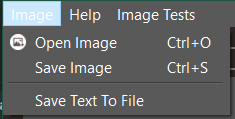
# 6. Apmācītā neironu tīkla izmantošana rakstīta teksta atpazīšanai

Lietotāja interfeisa izskats:



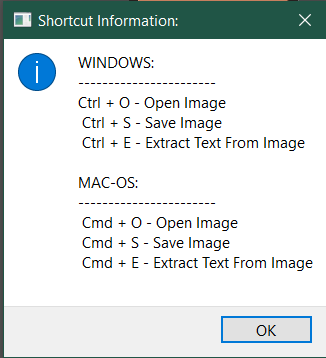
28. attēls: Lietotāja interfeisa pamatlogs, uzsākot lietotnes darbību

Izvēle atvērt attēlu, saglabāt pārveidoto input attēlu un saglabāt atpazīto tekstu:



29. attēls: Pārveidotā attēla saglabāšana

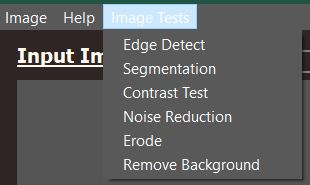
Lietotnes īsceļi abām platformām:



30. attēls: Platformu īsceļi

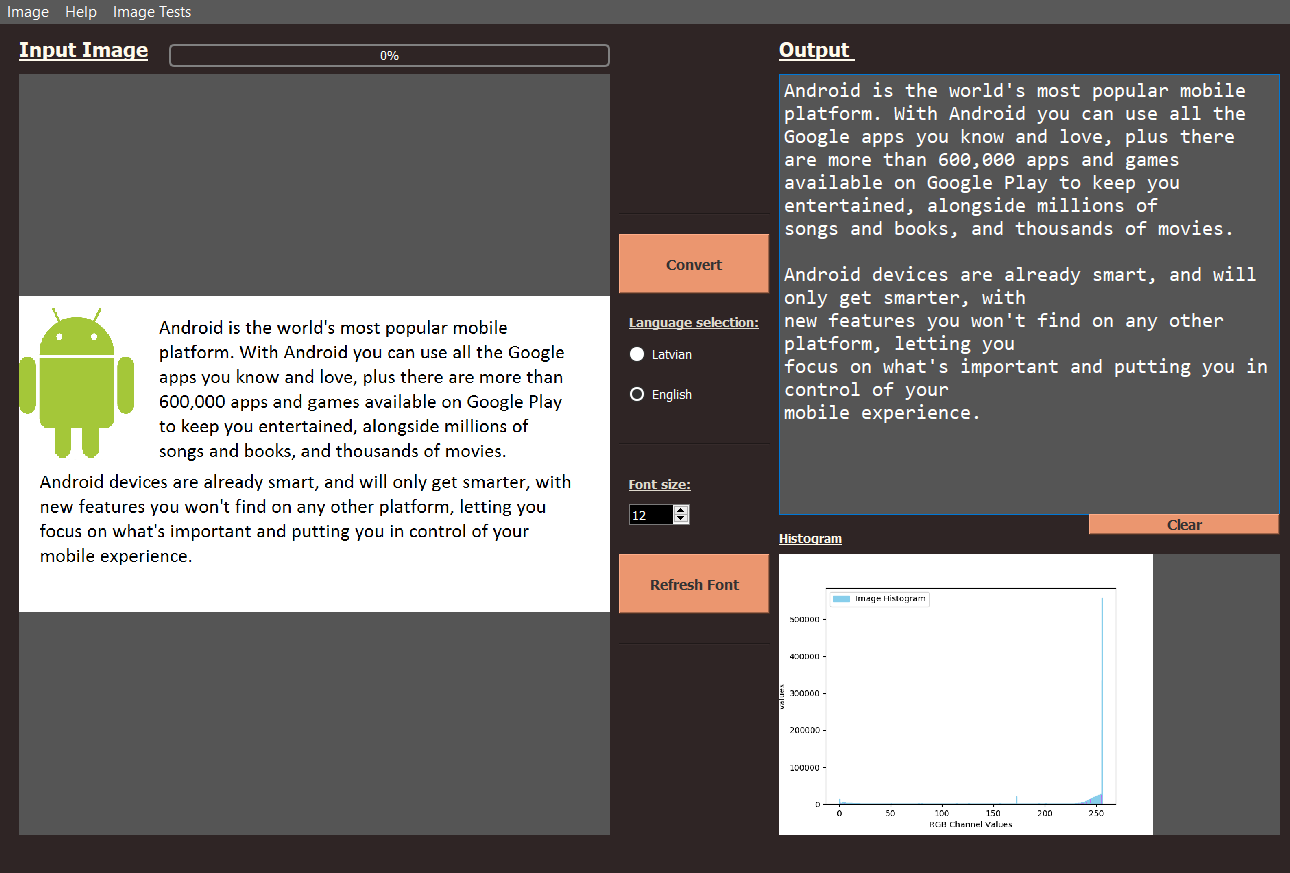
Iespēja veikt papildus attēlu apstrādi:

Lietotājs papildus funkcijas var izmantot, ja ir ievietots kāds “īpašs” vai “ārpus normas esošs” attēls, kuru tesseract programmatūra vēl aizvien nespēj nosalīt pēc iebūvētās standarta attēla apstrādes metožu izpildes.

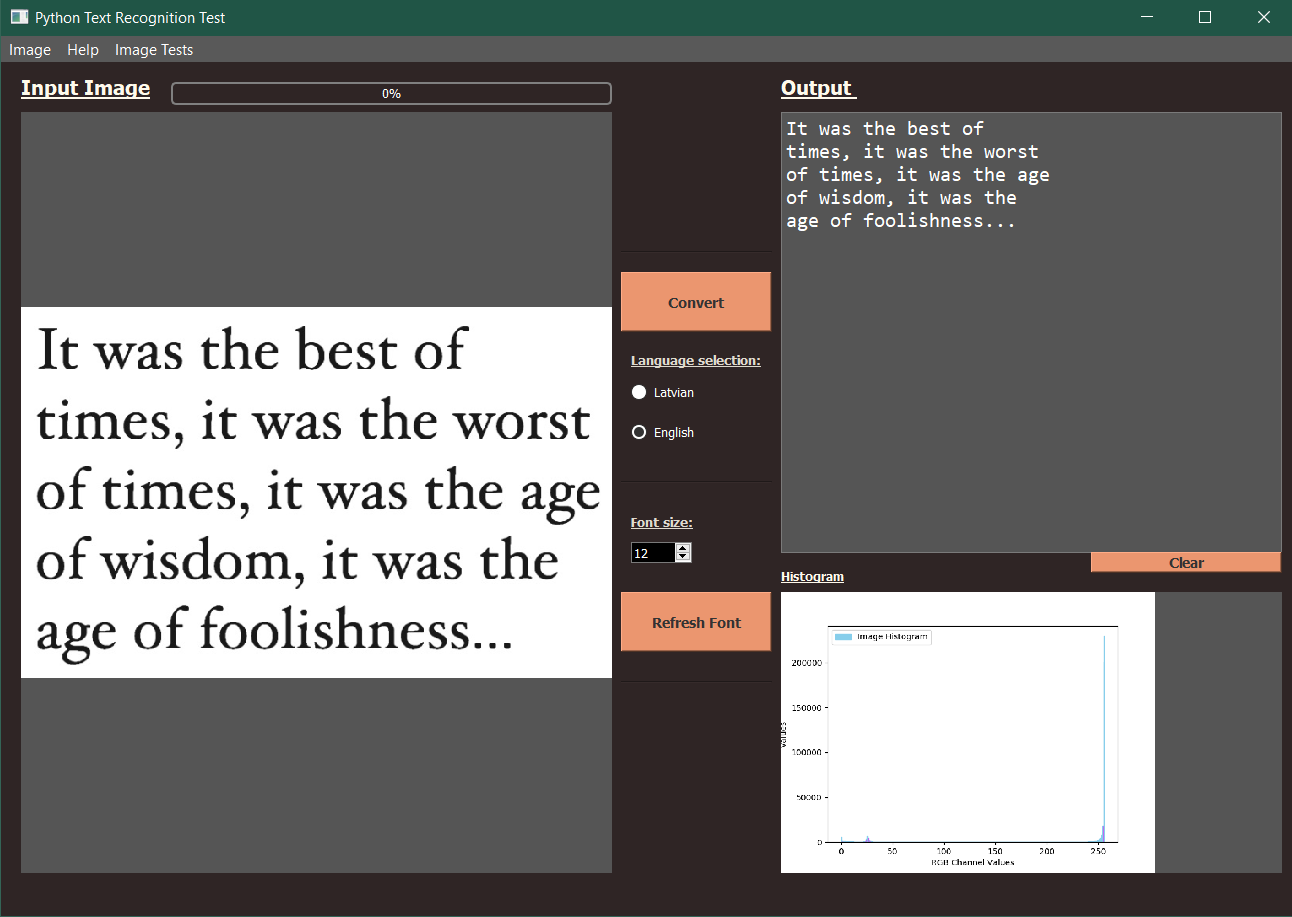


31. attēls: Papildus attēla apstrādes komandas

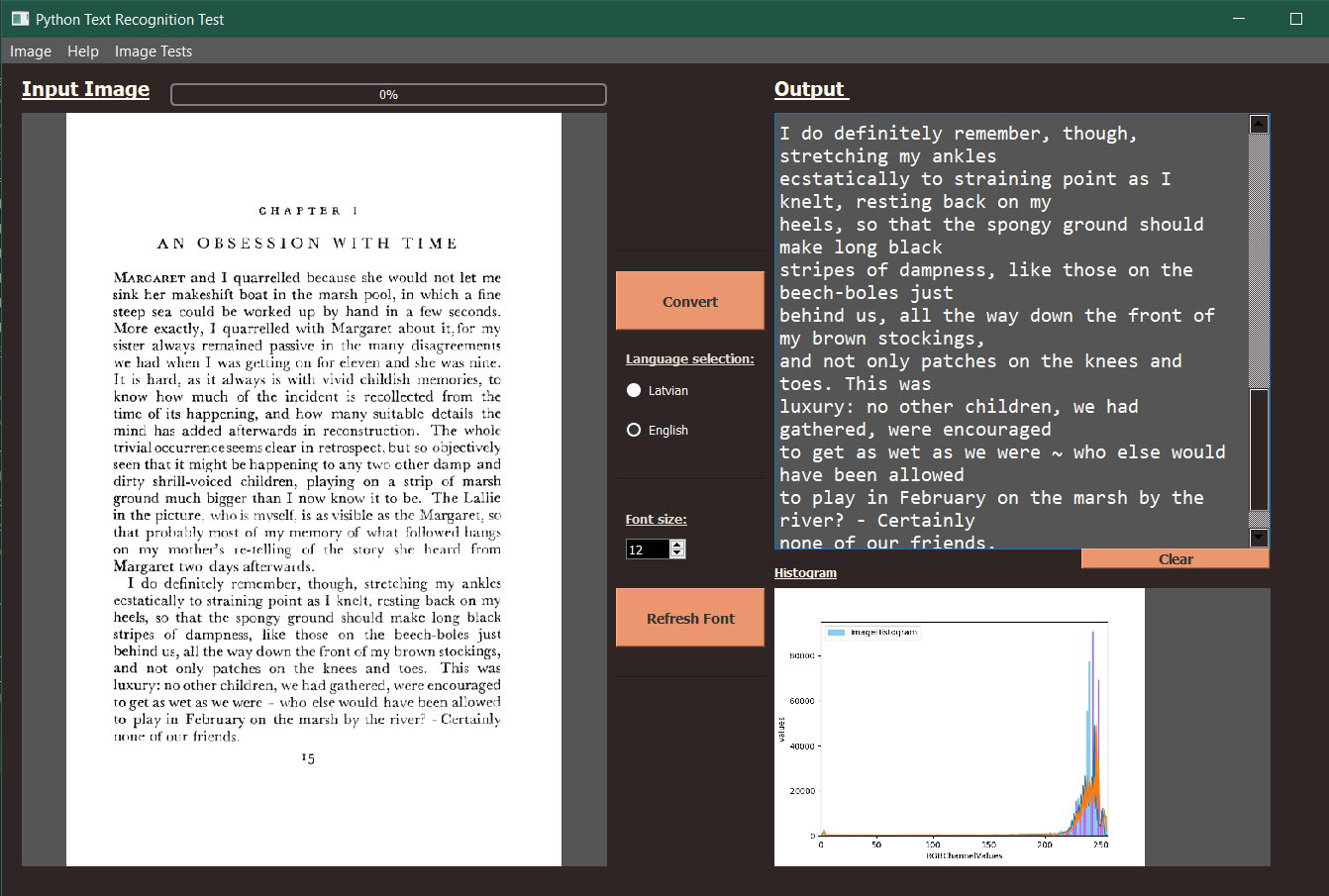
Piemērs tam kā tas darbojas:



32. attēls: Sistēmas darbošanās piemērs (1)



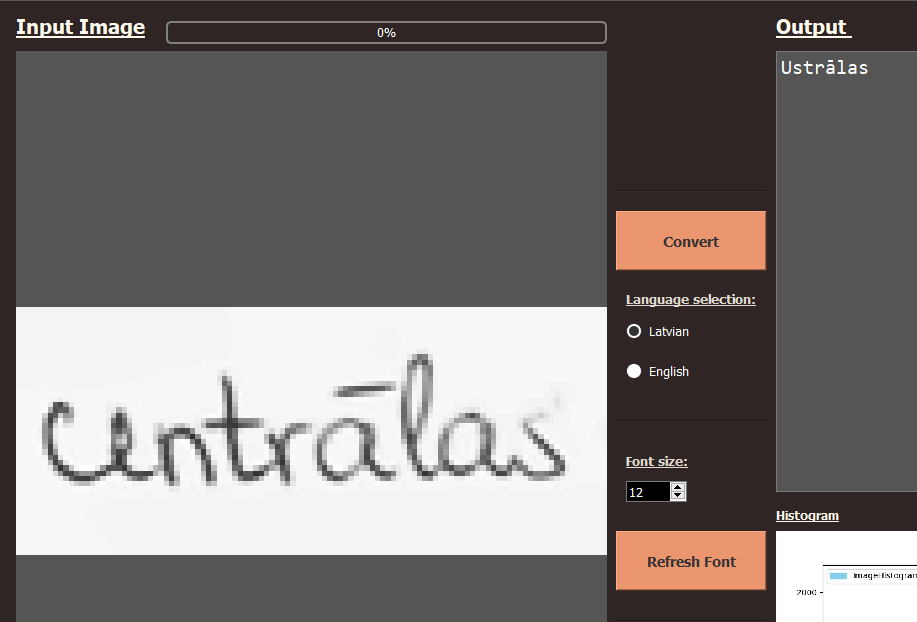
33. attēls : Sistēmas darbošanās piemērs (2)



34. attēls: Sistēmas darbošanās piemērs (3)



35. attēls: Sistēmas darbošanās piemērs (4)



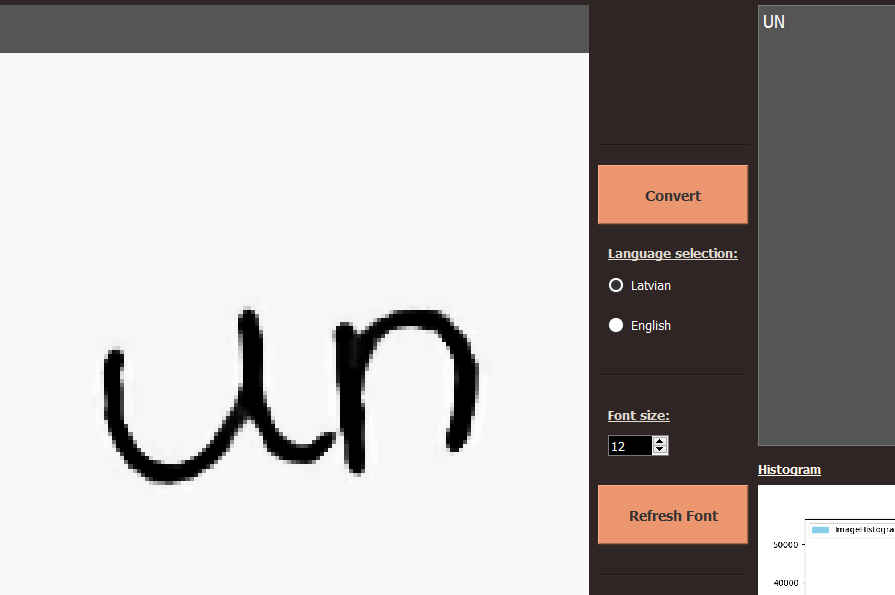
36. attēls: Sistēmas darbošanās piemērs (5)



37. attēls: Sistēmas darbošanās piemērs (6)



38. attēls: Sistēmas darbošanās piemērs (7)



39. attēls: Sistēmas darbošanās piemērs (8)**Secinājumi**

Augstas precizitātes rakstīta teksta apmācībai nepieciešams ļoti liels skaits ar apmācības datiem (aptuveni 100 000 attēlu)*.* Diemžēl kursa darba laikā nebija iespējams izveidot pietiekoši lielu attēlu datu bāzi pilnvērtīgai neironu tīkla apmācībai.

Teksta apazīšanas precizitāte ir atkarīga no attēla kvalitātes. Nepareizi nolasītus rādījumus var izraisīt sekojošās situācijas:

* Miglaini vai neskaidri attēli;
* Ar roku rakstīts teksts vai teksts slīprakstā;
* Mākslinieciski tekta fonti;
* Mazs teksta izmērs;
* Sarežģīts attēla fons, izteiktas ēnas, spīdums vai atspulgs uz teksta, vai perspektīva vai fokusa izmaiņa;
* Pārāk lieli vai neesoši lielie burti vārdu sākumos;
* Apakšteksts, Virsteksts vai nosvītrots teksts.

Darba laikā tika secināts, ka daudz efektīvāk ir sistēmu apmācīt nevis uz veseliem vārdiem, bet gan uz atsevišķu burtu atpazīšanu. Šis secinājums palīdzēja samazināt trenējamo datu apjomu.

Salīdzinot apmācītā modeļa “nas.traineddata” un Tesseract piedāvātā latviešu valodas “lav.traineddata” modeļa darbību uz rokraksta attēliem, bija iespējams novērot, atšķirīgus rezultātus. Apmācītais modelis, labāk spēja noteikt burtus “T”, “R”, “E”. Abiem modeļiem problemātiski bija atpazīt un atšķirt burtus “O” un “A”, “N” un “M”, “K” un “R”.