**RĪGAS TEHNISKĀ UNIVERSITĀTE**
Datorzinātnes un informācijas tehnoloģijas fakultāte
Lietišķo datorsistēmu institūts
Mākslīgā intelekta un sistēmu inženierijas katedra

**Oļegs Korsaks**
Bakalaura studiju programmas „Datorsistēmas”
students, stud. apl. nr. 051RDB146

# SALĪDZINOŠĀ ANALĪZE DATU KOPU FORMĀTIEM PYTORCH ATTĒLU KLASIFIKĀCIJAS UZDEVUMIEM

**Atskaite par bakalaura darbu**

Zinātniskais vadītājs
Dr.sc.ing, Pētnieks
**ĒVALDS URTĀNS**

Rīga 2023

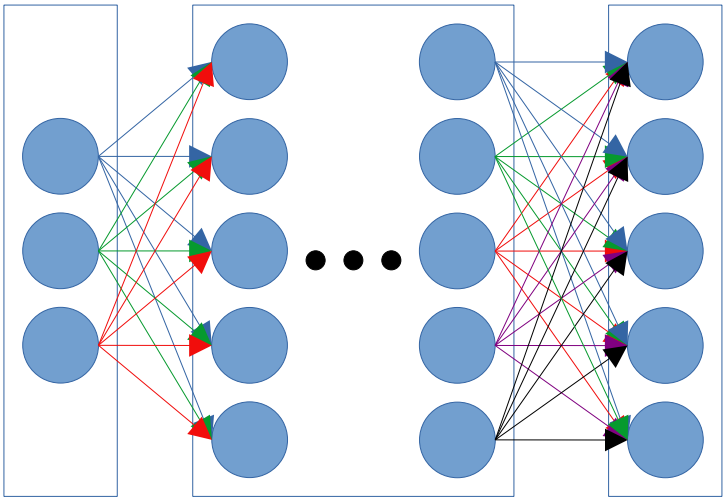
# Saturs

# 1 Ievads

## 1.1 Dziļā mašinmācīšanās

### *Pamata arhitektūras*

Makslīgais neironu tīkls sastāv no viena ieejas slāņa, viena vai dažiem slēptiem slāņiem, ka arī viena izejas slāņa. Katrs slānis satur vienu vai vairākus neironus. Tie ir saistīti ar blakusslāņu neironiem.



*Attēls 1: Makslīga neironu tīkla arhitektūras piemērs*

Eksistē tīkli, kuru neironi tiek pilnīgi saistīti ar visiem neironiem blakusslāņos, bet var būt saistīti tikai daļēji. Katrai saitei ir savs svars un nobīde. Tie ir apmācamie parametri. Apmācīšanas procesa mērķis ir atrast tādus svarus un nobīdes visām saitēm, ar kuriem tīkls veidotu pareizus rezultātus izejas slānī.

### Linārie slāņi

Stingras formāla teorija par to, kā atlasīt neironu tīkla slāņus un konfigurāciju neeksistē, un, lai gan vienīgais veids, kā noregulēt dažus hiperparametrus, ir tikai mēģinājumu un kļūdu metode (piemēram, meta-apmācība), taču joprojām pastāv dažas heiristikas, vadlīnijas un teorijas, kas joprojām var palīdzēt ievērojami samazināt piemērotu arhitektūru meklēšanas telpu.

### Lineārais nobīdes slānis

$$y=b$$

Šis slānis būtībā iemācās konstanti. Tas spēj apgūt nobīdi, novirzi, slieksni vai vidējo vērtību. Ja neironu tīklu izveidot tikai no šī slāņa un apmācit to, izmantojot datu kopu, vidējās kvadrātiskās kļūdas zudums liks šim slānim konverģēt uz izejas vidējo vērtību.

Piemēram, ja ir šāda datu kopa {1, 1, 2, 2, 3, 3} un neironu tīkls tiek spiests to saspiest līdz unikālai vērtībai $b$, loģiskākā konverģence būs ap vērtību $b=2$ (kas ir datu kopas vidējais lielums, lai samazinātu zaudējumus līdz maksimumam. Jebkura vērtība, kas pārsniedz šo nobīdi, būs pozitīva, jebkura vērtība, kas ir zemāka par šo nobīdi, būs negatīva. Tas ir tāpat kā pārdefinēt, no kurienes jāsākas nobīdei 0.

## Lineārais slānis

$$y=W\cdot x$$

Lineāra funkcija bez nobīdes ir spējīga iemācīties vidējo korelācijas koeficientu starp ievadu un izvadu. Piemēram ja $x$ un $y$ pozitīvi korelē - $W$ vērtība būs pozitīva. Un otrādi. Gadījumā, kad $x$ un $y$ ir savstarpēji neatkarīgi - $W$ būs vienāds ar 0.

Vēl viens veids, kā uztvert šo slāni: Ieviest jaunu mainīgo $A=\frac{y}{x}$ un izmantot “novirzes slāni” no iepriekšējās sadaļas, tas iemācīsies vidējo A. (kas ir vidēja izvades/ievades attiecības rādītājs, tādējādi vidējais koeficients, kurš nosaka, cik ātri izvade mainās attiecībā pret ievadi).

## Lineārais padeves slānis

$$y=W\cdot x+b$$

Lineārais padeves slānis ir parasta lineāra slāņa un nobīdes kombinācija. Tas ir spējīgs iemācīties nobīdi un korelācijas koeficientu. Matemātiski tas definē taisnes vienādojumu.

## Lineāro slāņu ierobežojumi:

- Visi trīs lineāro slāņu tipi var iemācīties tikai lineāras attiecības.
- Šo slāņu sakraušana uzreiz vienu pēc otra ir pilnīgi bezjēdzīga un noved uz skaitļošanas resursu izšķiešanu:

Pieņemot, ka ir divi secīgi lineārie padeves slāņi $y_1$ un $y_2$ :

$$y_1=w_1\cdot x+b_1$$
$$y_2=w_2\cdot y_1+b_2$$

Var pārrakstīt $y_2$ sekojoši:

$$y_2=w_2\cdot(w_1 x+b_1)+b_2$$
$$y_2=(w_2\cdot w_1)x+(w_2\cdot b_1+b_2)$$
$$y_2=w\cdot x+b$$

To pašu var izdarīt ar jebkuru secīgi izvietotu lineāro slāņu skaitu. Viens lineārais slānis ir spējīgs reprezentēt jebkuru secīgi izvietotu lineāro slāņu skaitu.

## Aktivizācijas funkcijas

### Sigmoid

Šī funkcija saspiež ieejas vektoru (0, 1) robežās un tiek pielietota katram vektora elementam atsevišķi.

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

Piemēram [ 35.4, 10.1, -4.2, 0 ] tiks saspiests uz [ 0.9999999999999577, 0.999958922132235237, 0.0147740316932730578, 0.5 ]

### ReLU

Tradicionāli neironu tīklos tiek izmantotas dažas izplatītas nelineāras aktivizācijas funkcijas, piemēram, sigmoīdās funkcijas un hiperboliskais tangenss, lai iegūtu katram neironam atbilstošās aktivizācijas vērtības. Nesen tā vietā sāka izmantot ReLu funkciju[1], lai aprēķinātu aktivizācijas vērtības tradicionālajās neironu tīklu vai dziļo neironu tīklu paradigmās.

$$f(x)=\begin{cases} x, x\geq 0 \\ 0, x<0 \end{cases}$$

Iemesli sigmoīdā un hiperboliskā tangensa aizstāšanai ar ReLu ir šādi:

1. Aprēķinu taupīšana — ReLU funkcija spēj paātrināt dziļo neironu tīklu apmācības ātrumu salīdzinājumā ar tradicionālajām aktivizācijas funkcijām, jo ReLU atvasinājums ir 1 pozitīvai ievadei. Sakarā ar pastāvīgu, dziļu neironu tīklu apmācību fāzē nav nepieciešams papildu laiks kļūdas aprēķināšanai.
2. Izzūdoša gradienta problēmas risināšana - ReLU funkcija neizraisa izzūdoša gradienta problēmu, kad slāņu skaits pieaug. Tas ir iespējams tāpēc, ka šai funkcijai nav asimptotiskas augšējās un apakšējās robežas. Tādējādi agrākais slānis (pirmais slēptais slānis) spēj uztvert kļūdas, kas nāk no pēdējiem slāņiem, lai pielāgotu visus svarus starp slāņiem. Turpretim tradicionālā aktivizēšanas funkcija, piemēram, sigmoīds, ir ierobežota no 0 līdz 1, tāpēc pirmajā slēptajā slānī kļūdas kļūst mazas. Šis scenārijs var novest pie slikti apmācīta neironu tīkla.

### ELU

Eksponenciālā lineārā vienība. Šī aktivizācija funkcija paātrinā apmācību un to precizitāti.

$$f(x)=\begin{cases} x, x\geq 0 \\ \alpha(e^{x}-1), x<0 \end{cases}$$

Priekšrocības:

- Mēdz ātrāk saplūst nekā ReLU (jo vidējās ELU aktivizācijas ir tuvāk nullei)
- Labāka vispārināšanas veiktspēja nekā ReLU
- Pilnībā nepārtraukta funkcija
- Pilnībā diferencējams funkcija
- Nav izzūdoša gradienta problēmas
- Nav sprāgstoša gradienta problēmas

- Nav dead ReLU problēmas
- Joprojām ir iespēja izmantot kā ReLU, pieņemot $\alpha = 0$

Trūkumi:

- Lēnāks aprēķins (nelinearitātes dēļ priekš negatīvām vērtībām)

### Softmax

Softmax ir funkcija, kas arī saspiež ieejas vektoru (0, 1) robežās un kuras rezultējušo elementru summa ir 1. Elementu uzskata pēc klasmēm un to vērtības pēc klašu varbūtībām[2]. Šī funkcija strādā ar visiem ieejas vektora elementiem priekš katra izejas rezultāta elementa aprēķināšanai, jo ir atkārība no elementu summas. Priekš atsevišķas klases $x_i$ Softmax funkcija izskatas šādi:

$$f(x)_i = \frac{e^{x_i}}{\sum_{j}^{C} e^{x_j}}$$

Kur $x_j$ ir reitingi, kuri tika saņemti no neirontīkla katrai klasei iekš C vektora. Var redzēt, ka Softmax aktiviācija katrai klasei ir atkārīga no visiem reitingiem.

$$L(y, y') = -\frac{1}{N} \sum y \cdot \log(y')$$

$$\frac{\partial L(y, y')}{\partial y'} = -y \cdot \frac{1}{y'} = -\frac{y}{y'}$$

$$SoftMax(y = j|x) = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}$$

## Kļūdas funkcijas

Kļūdas funkcija palīdz noteikt cik tālu tekoša prognozēta vērtība ir no patiesas. Un ja to pielietot visiem datu eksemplāriem – ar to var noteikt, cik labi tekošais modelis var prognozēt rezultātus kopumā. Ideālā gadījumā kļūdai jābūt vienādai nullei, gan apmācības datu eksemplāriem, gan pārbaudes. Tātad apmācība cenšas kļūdu samazināt.

### MAE

Mean absolute error vai vidēja absolūta kļūda:

$$L_{MAE} = \frac{1}{N} \cdot \sum_{i=0}^{N} \left| (h_\theta(x_i) - y_i) \right|$$

Tā ir vidēja absolūta starpība starp pareizas un prognozētas vertībām. Kļūda pieaug lineāri un tiek pielietota regresijas uzdevumiem, kuru rezultāts ir viena vērtība.

## MSE

Mean squared error vai vidēja kvadrātiskā kļūda:

$$L_{MSE}=\frac{1}{N}\cdot\sum_{i=0}^{N}(h_\theta(x_i)-y_i)^2$$

Tā ir vidēja starpība starp pareizas un prognozētas vertībām, kas tiek pacelta kvadrātā. Kvadrāts palīdz izvairīties no negatīvām vērtībām. Kā arī kļūdas vērtība pieaug straujāk, salīdzinot ar MAE. Tiek pielietota regresijas uzdevumiem.

## CCE

Categorical cross-entropy vai kategoriskā krustentropija ir zuduma funkcija, kura tiek pielietota vairāku klašu klasifikācijas uzdevumos. Šajos gadījumos katrs datu kopas eksemplārs var piederēt tikai vienai klasei no vairākām. Neirontīkla modelim ir jānoteic, kurai tieši. Ar to palīdzību var kvantitātīvi noteikt starpību diviem varbūtības sadalījumiem.

$$zudums=-\sum_{i=1}^{n}y_i\cdot\log y'_i$$

Kur $y'_i$ ir skalāra vērtība no modeļa rezultējoša vektora un $y_i$ ir attiecīga skalāra vērtība no mērķa vektora. $n$ - kopējais elementu skaits rezultējoša vektorā.

## BCE

Binary cross-entropy vai binārā krustentropija ir līdzīga CCE, bet tā var noteikt eksemplāra piederību vairākām klasēm[3]. Lai tas būtu iespējams, rezultējošo vektoru aktivizē ar Sigmoid funkciju, katru elementu atsevišķi.

$$zudums=-\frac{1}{n}\sum_{i=1}^{n}y_i\cdot\log y'_i+(1-y_i)\cdot\log(1-y'_i)$$

Kur $y'_i$ ir skalāra vērtība no modeļa rezultējoša vektora un $y_i$ ir attiecīga skalāra vērtība no mērķa vektora. $n$ - kopējais elementu skaits rezultējoša vektorā.

## *Atpakaļizplatīšanās algoritms*

Viena slāņa neironu tīklā apmācības process ir salīdzinoši vienkāršs, jo kļūdu (vai zaudējuma funkciju) var aprēķināt kā tiešu svaru funkciju, kas ļauj viegli aprēķināt gradientu. Daudzslāņu tīklu gadījumā problēma ir tāda, ka zudums ir sarežģīta kompozīta funkcija no iepriekšējo slāņu svariem. Kompozīta funkcijas gradients tiek aprēķināts, izmantojot atplakaļizplatīšanās algoritmu. Atplakaļizplatīšanās algoritmam ir divas galvenās fāzes. Tiešā fāze ir nepieciešama, lai aprēķinātu izvades vērtības un vietējos atvasinājumus dažādos mezglos, un atpakaļgaitas fāze ir nepieciešama, lai uzkrātu šo vietējo vērtību produktus visos ceļos no mezgla līdz izvadei:

1. Tiešā fāze: šajā fāzē apmācības instances ievade tiek ievadīta neironu tīklā. Tā rezultātā tiek veikta aprēķinu kaskāde uz priekšu pa slāņiem, izmantojot pašreizējo svaru kopu. Pēc tam

notiek paredzama izvada salīdzināšana ar apmācības rezultātu, un tiek aprēķināts zaudējuma funkcijas atvasinājums attiecībā uz izvadi. Šī zaudējuma atvasinājums ir jāaprēķina attiecībā uz svariem visos slāņos atpakaļgaitas fāzē.

2. Atpakaļgaitas fāze: galvenais mērķis ir iegūt zaudējuma funkcijas gradientu attiecībā pret dažādiem svariem, izmantojot diferenciālu ķēdes likumu. Šie gradienti tiek izmantoti, lai atjauninātu svarus. Tā kā šie gradienti tiek iegūti atpakaļ virzienā, sākot no izvades mezgla, šis mācību process tiek saukts par atpakaļejošu fāzi.

Šīs algoritms cenšas mainīt tīkla svarus un nobīdes tā, lai kļūda būtu 0.

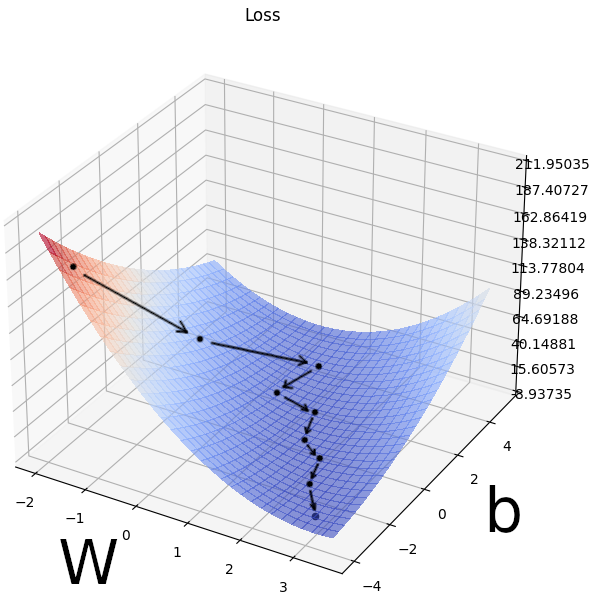
$$\theta_0 := \theta_0 - \alpha \cdot \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$b := b - \alpha \cdot \frac{\partial}{\partial b} J(b, W)$$

$$\theta_1 := \theta_1 - \alpha \cdot \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$W := W - \alpha \cdot \frac{\partial}{\partial W} J(b, W)$$

Pieņemsim, ka modelis ir:

$$y' = M(x) = Linear(W_1, b_1, W_2, b_2, x) = Linear(W_2, b_2, ReLU(Linear(W_1, b_1, x)))$$
$$Linear(W_i, b_i, x_i) = W_i \cdot x_i + b_i$$
$$ReLU(x_i) = \begin{cases} x_i, x_i \geq 0 \\ 0, x_i < 0 \end{cases}$$

Kur, piemēram, $J(\theta_0, \theta_1) = L_{MAE} = \frac{1}{N} \cdot \sum_{i=0}^{N} (h_\theta(x_i) - y_i)$ ir MAE kļūdas funkcija un $\alpha$ ir apmācības koeficients, kurš noteic, cik strauji svars $W$ un nobīde $b$ tiek mainīti. Kļūdas funkcijas atvasinājums noteic vai parametru ir jāpalielina, vai jāsāmazina un uz kādu lielumu.




*Attēls 2: Kļūdas minimizācijas process*

## *Apmācāmo parametru optimizācijas algoritmi*

### Stohastiskā gradienta nolaišanās algoritms (SGD)

Šī algoritma mērķis ir mainīt nobīdi(es) $\theta_0, b$ un svaru(s) $\theta_1, W$ lai minimizētu zuduma funkcijas rezultātu

$$\theta_0 := \theta_0 - \alpha \cdot \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$b := b - \alpha \cdot \frac{\partial}{\partial b} J(b, W)$$

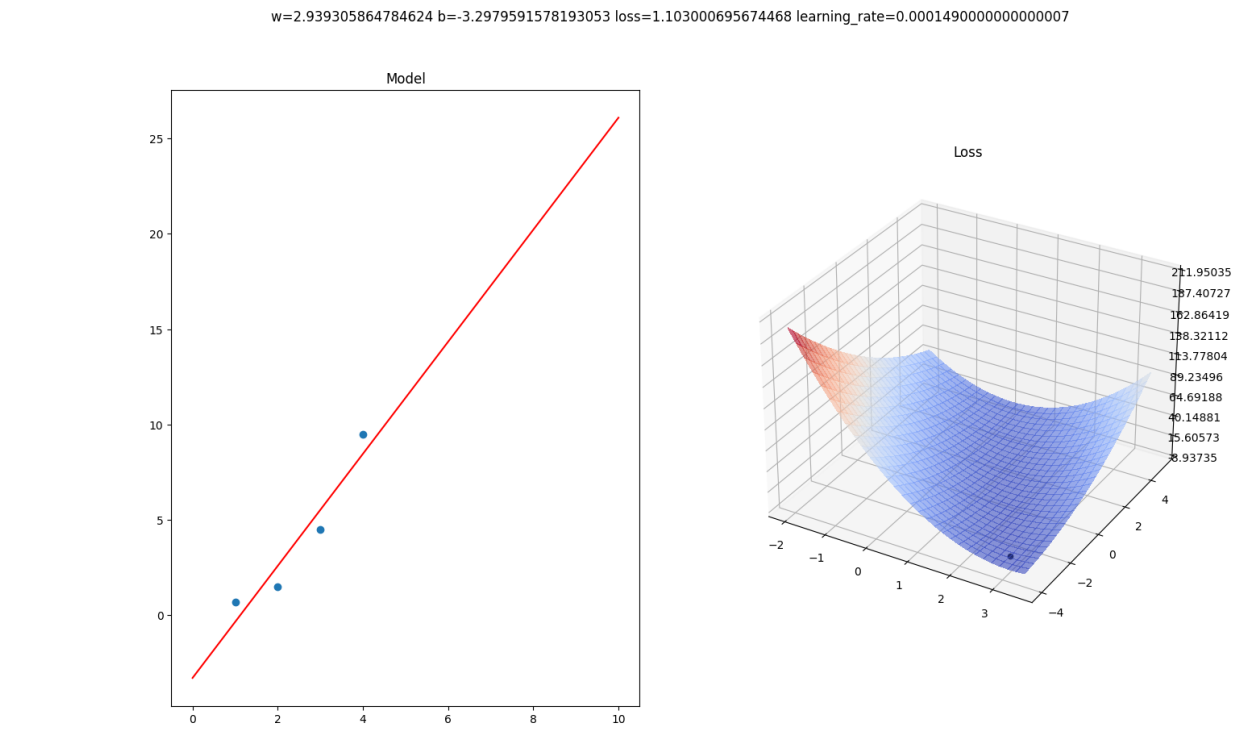
$$\theta_1 := \theta_1 - \alpha \cdot \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$W := W - \alpha \cdot \frac{\partial}{\partial W} J(b, W)$$

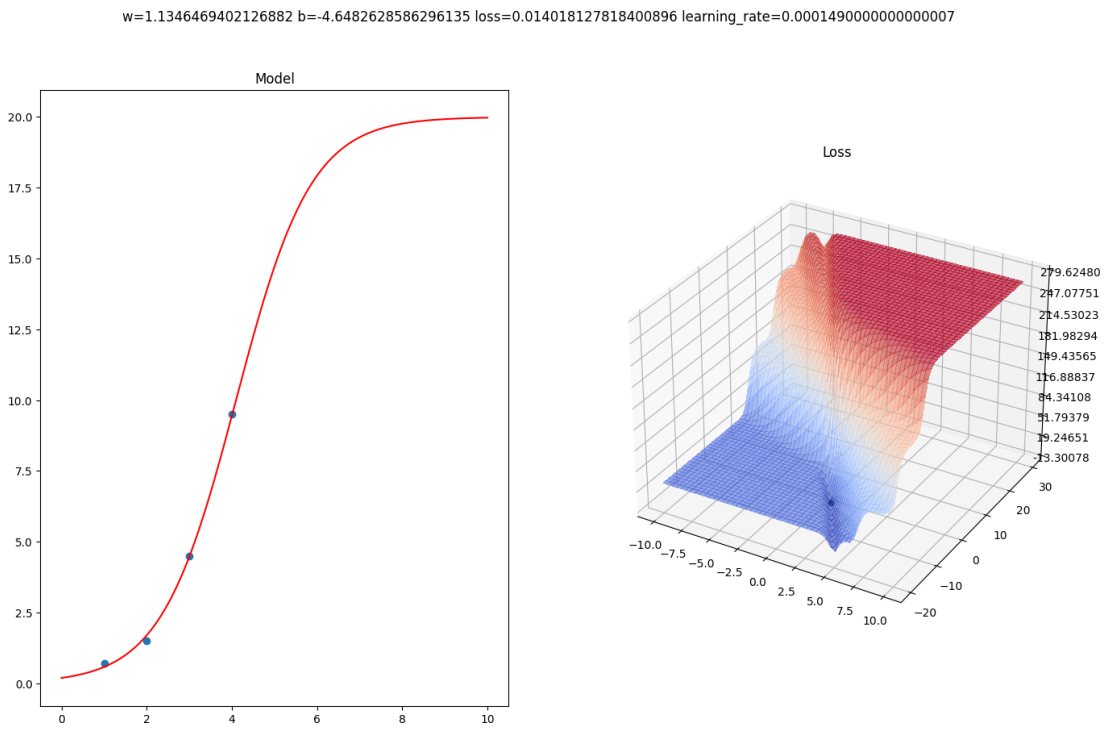
bet lai vienlaicīgi izmainītu abus mainīgus – kodā ir nepieciešams izmantot pagaidu mainīgos, citādi viena parametra maiņa ietekmēs citu.

$\alpha$ - ir apmācības ātrums, kurš noteic cik atri $b$ un $W$ mainīsies.

Zuduma funkcijas daļējs atvasinājums būs pozitīvs slīpums (atgriezīs pozitīvu skaitli), ja zudums (kļūda) būs lokālo minimumu labajā pusē, kā rezultātā no W (vai b) tiks atņemta slīpuma vērtību, kas reizināta ar mācīšanās ātrumu. Pretējā gadījumā tā tiks pievienota. Ideālā situācijā, kad zudums ir 0, nekādas izmaiņas nav jādara un var teikt, ka modelis ir apmācīts.




*Attēls 3: Zuduma plakne priekš lineāra modeļa*




*Attēls 4: Zuduma plakne lineāram modelim ar sigmoida aktivizācijas funkciju*

Kubiskajai funkcijai $b+W\cdot x^3$ bez aktivizācijas funkcijas, laba lieta šajā gadījumā zuduma funkcijas atvasinājums $dW, db$ ir tas pats kā lineārajai funkcijai. Un $dx$ ir:

$$dx\,cubic: \frac{\partial}{\partial x}[W\cdot x^3+b]=W\cdot 3\cdot x^2$$

## RMSprop

Stohastiskā gradienta nolaišanas algoritmam ir potenciāla problēma – apmācības koeficients ir konstants un visiem parametriem ir viens un tas pats. Tas derētu viegliem uzdevumiem, kuriem kļūdas virsma ir glūda. Taču gadījumos, kad neironu tīkls ir sarežģīts un kļūdas virsma arī ir sarežģīta, gradients var vai nu pazust, vai nu "eksplodēt".

$$v_{dw}=\beta\cdot v_{dw}+(1-\beta)\cdot dw^2$$
$$v_{db}=\beta\cdot v_{dw}+(1-\beta)\cdot db^2$$
$$W=W-\frac{\alpha\cdot dw}{\sqrt{v_{dw}}+\epsilon}$$
$$b=b-\frac{\alpha\cdot db}{\sqrt{v_{db}}+\epsilon}$$

Situāciju var uzlabot adaptīvs apmācības koeficients, kas nozīmē, ka viņš mainās ar laiku un vairs neskaitās par hiperparametru. Šīs algoritms samazina soli lielam gradientam lai izvairītos no “eksplozijas” un palielina soli mazam gradientam lai izvairītos no pazušanas.

### *Metrikas*

Lai varētu saprast, cik labi modelis funkcionē, eksistē dažāda veida metrikas.

## Apjukuma matrica

| | | Minējums | |
|---|---|---|---|
| | | Pozitīvs | Negatīvs |
| Patiesība | Pozitīvs | TP | FN |
| | Negatīvs | FP | TN |

Divu variantu gadījumā var sastādīt tabulu, kur **T*** ir situāciju skaits, kad minējums sakrīt ar patieso atbildi. Un **F*** ir situāciju skaits, kad minējums nesakrīt.

## Akurātība

Tā ir pareizo minējumu skaita pret kopēju minējumu skaitu attiecība:

$$akurātība = \frac{pareizo\,minējumu\,skaits}{kopējais\,minējumu\,skaits} = \frac{TP+TN}{TP+TN+FP+FN}$$

Šo metriku var izmantot gadījumos, kad dati ir sabalansēti. Piemēram, ja modelis neko neprognozē, bet tikai atgriež vienu un to pašu atbildi, kuras biežums ir 95%, tad sanāks, ka modelim akurātība ir 0.95. Kas nav taisnība un visos citos gadījumos modelis nekad neatgriezīs pareizo atbildi.

## Precizitāte

Ideālā gadījumā precizitātei jābūt 1, ja klasifikācija notiek labi. Tas notiek tad, kad pareizo minējumu skaits ir vienāds pareizo un nepareizo pozitīvo minējumu skaitam.

$$precīzitāte = \frac{TP}{TP+FP}$$

Gadījumā, kad nepareizi pozitīvo (FP) minējumu skaits sāk pieaugt – precizitātes vērtība samazinās.

## Recall

Labam klasifikātoram Recall arī jābūt vienādam 1. Bet to skaita kā pareizo minējumu attiecību pret pareizo pozitīvo un nepareizo negatīvo minējumu skaitu. Jo mazāks nepareizo negatīvo minējumu, jo labāks Recall (tuvāk 1).

$$recall = \frac{TP}{TP + FN}$$

Ja viltus negatīvo minējumu (FN) skaits pieaugs, tad Recall vērtība samazināsies.

### F1 reitings

F1 reitings ir nepieciešams, ja ir svarīga precizitāte. Ir jau iepriekš noskaidrots, ka precizitāte nozīmē pareizi identificēto pozitīvo un negatīvo minējumu procentuālo daļu.

$$F1\,reitings = 2 \cdot \frac{precīzitāte \cdot recall}{precīzitāte + recall}$$

Pieņemot lēmumus, ir tendence nekoncentrēties uz daudzām dažādām lietām, turpretim viltus pozitīvajiem un viltus negatīvajiem gadījumiem. F1 reitings var būt labāks rādītājs[4].

## *Hiperparametri*

Hiperparametri ir definēti kā parametri, kuri tiek skaidri definēti, lai kontrolētu apmācīšanas procesu. Šie parametri izsaka modeļa "augsta līmeņa" īpašības, piemēram, tā sarežģītību vai to, cik ātri tam vajadzētu mācīties[5]. Hiperparametri parasti tiek fiksēti pirms faktiskā apmācības procesa sākuma. Hiperparametrus var iedalīt divās kategorijās:

### Optimizātora hiperparametri

Tie ir mainīgie vai parametri, kas vairāk saistīti ar optimizācijas un apmācības procesu, nevis ar pašu modelēšanu. Šie parametri palīdz noregulēt vai optimizēt modeli pirms faktiskā apmācības procesa sākuma, lai sāktu apmācības procesu pareizajā vietā:

- Apmācības ātrums: tas ir hiperparametrs, kas kontrolē, cik strauji neironu tīkla svari tiek pielāgoti attiecībā pret gradientu.
- Mini-partijas lielums: tas ir hiperparametrs, kas ietekmē apmācības resursu prasības, kā arī apmācības ātrumu un iterāciju skaitu.
- Apmācības atkārtojumu vai epohu skaits: Epoha - tā ir viena pilnīga apmācības datu iziešana.

### Modeļa hiperparametri

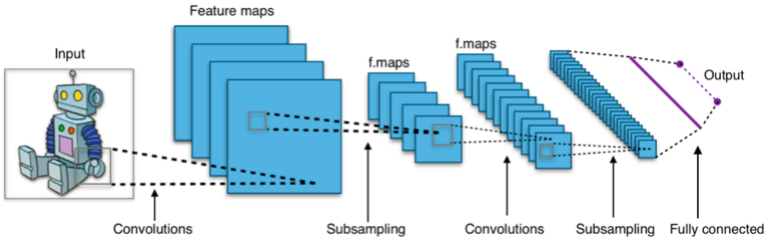
Tie ir mainīgie, kas ir vairāk iesaistīti modeļa arhitektūrā vai struktūrā. Tas palīdz definēt modeļa sarežģītību, pamatojoties uz:

- Slāņu skaits
- Slēptās vienības: slēpto slāņu skaits neironu tīklā, jo sarežģītāks modelis (nozīmē vairāk slēpto slāņu), jo lielākas mācīšanās spējas modelim būs nepieciešamas.

## 1.2 Attēlu klasifikācija

### Konvolūcijas tikls (ConvNet)

Konvolucionālajiem neironu tīkliem ir vairāki pielietojumi objektu noteikšanā, lokalizācijā, video un teksta apstrādē. Daudzi no šiem pielietojumiem darbojas pēc konvolucionālo neironu tīklu izmantošanas pamatprincipa, lai nodrošinātu inženierijas funkcijas, kurām papildus var izveidot daudzdimensiju pielietojumu[5].

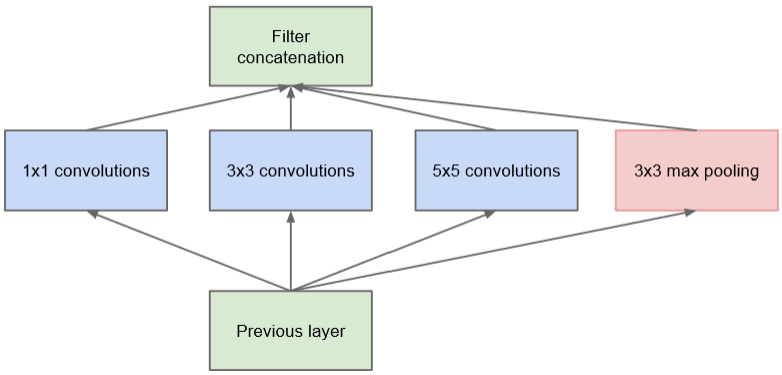



*Attēls 5: Tipisks konvolūcijas tīkls*

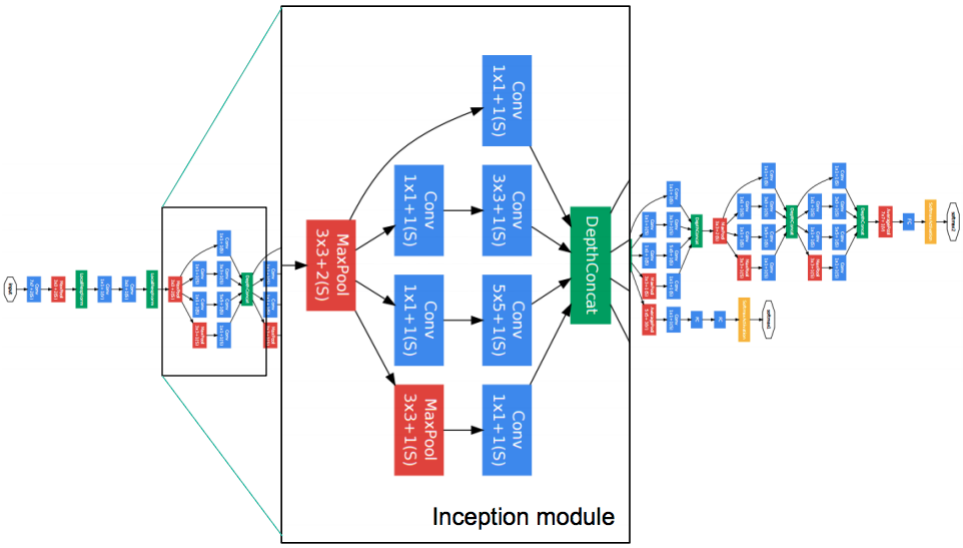
Konvolucionālo neironu tīklu panākumi joprojām ir nepārspēti gandrīz nevienai neironu tīklu klasei. Pēdējos gados pat ir ierosinātas konkurētspējīgas metodes sequence-to-sequence apmācīšanai, kas tradicionāli ir bijusi rekurento tīklu joma.

### InceptionNet

Inception tīklu ideja sākas no pamata pieņēmuma, ka attēla objektiem ir dažādi mērogi. Attāls objekts var aizņemt nelielu attēla apgabalu, bet tas pats objekts, nonākot tuvāk, var aizņemt lielāko attēla daļu. Tas rada grūtības standarta konvolūcijas tīklam, kur neironiem dažādos slāņos ir fiksēts uztverošā lauka izmērs, kā noteikts ievades attēlam. Parasts tīkls var būt labs objektu detektors noteiktā mērogā, taču pretējā gadījumā tos var palaist garām.




*Attēls 6: Inception bloks*




*Attēls 7: Inception tīkla piemērs*

Lai atrisinātu šo problēmu, Szegedy et al ierosināja jaunu arhitektūru: tādu, kas sastāv no Inception blokiem. Inception bloks sākas ar kopīgu ievadi un pēc tam sadala to dažādos paralēlos ceļos (vai torņos). Katrs ceļš satur vai nu konvolūcijas slāņus ar dažāda izmēra filtru, vai arī apvienošanas slāni. Tādā veidā vieniem un tiem pašiem ievades datiem tiek izmantoti dažādi uztverošie laukus. Inception bloka beigās dažādu ceļu izejas tiek savienotas.

## 1.3 PyTorch vide

PyTorch ir optimizēta mašīnmācīšanas bibliotēka, kura atvieglo darbu gan ar procesoru (CPU), gan ar videokārti (GPU). Programēšanas valoda ir Python.

Modeļa definēšanas piemērs PyTorch vidē:

```
class Model(Module):
    def __init__(self):
        super().__init__()

        self.layers = Sequential(
            Linear(in_features=13, out_features=10, device=device),
            Tanh(),
            Linear(in_features=10, out_features=5, device=device),
            LeakyReLU(),
            Linear(in_features=5, out_features=1, device=device)
        )

    def forward(self, x):
        y_prim = self.layers.forward(x)

        return y_prim
```

PyTorch Datasets vs TFRecord ar dažiem piemēriem
Atšķirības starp frameworks

## Datu ielādes process

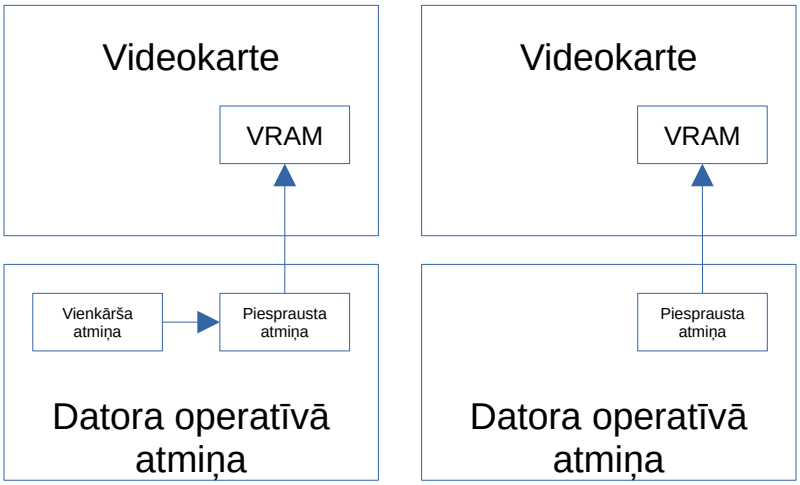
Datu ielādes process sastāv no datu kopas (Dataset) un datu ielādēja (DataLoader) klasēm. Dataset klase ir atbildīga par datu kopas izmēra noteikšanu caur **__len__()** metodi un pēc indeksa izvēlēta ieraksta atgriešanu caur **__getitem__(index)** metodi. Pašus datus var lejupielādēt no tīmekļa, vai ielādēt no diska inicializācijas solī.

```
class CustomDataset(torch.utils.data.Dataset):
    def __init__(self):
        super().__init__()
        self.x = [1, 2, 3, 4, 5, 6]
        self.y = [0, 1, 0, 1, 0, 1]

    def __getitem__(self, index):
        return self.x[index], self.y[index]

    def __len__(self):
        return len(self.x)
```

Pēc datu kopas definīcijas un inicializācijas to sadala uz apmācīšanas un pārbaudes kopām. Tālāk katru no tām padod datu ielādejiem. Tālāk darbs notiek ar datu ielādejiem.

```
data_loader_train = torch.utils.data.DataLoader(
    dataset=dataset_train,
    batch_size=BATCH_SIZE,
    shuffle=True,
    pin_memory=USE_CUDA,
    num_workers=WORKER_COUNT
)
```

Jo lielāka datu kopa ir, jo lielāki vajadzīgi: operatīva atmiņa (RAM), operatīva atmiņa videokartē (VRAM), kā arī vieta uz cieta diska. Un jo lielāki dati, jo vairāk tie pārvietosies no diska uz RAM un pēc tam uz VRAM.
Gadījumos, kad RAM/VRAM nav pietiekoši daudz, var izmantot **batch_size** lai vienlaikus ielādētu tikai noteikto datu eksemplāru skaitu. Lai paātrinātu datu ieladi, var arī izmantot **pin_memory=True**, lai datus ielādētu “piespraustā” CUDA atmiņā bez vienkāršas atmiņas starpniecības.



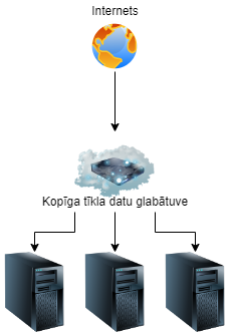
*Attēls 8: "Piespraustas atmiņas" darbības īpašība*

Gadījumā, ja datu partijas ielāde aizņem vairāk laika nekā to izmantošana apmācībai, tad to var ielādēt paralēli vairākos procesos (**num_workers**).

# 2 Metodoloģija

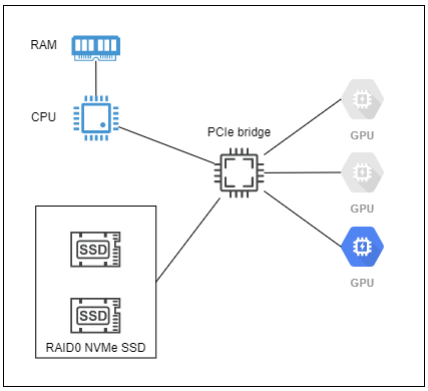
## 2.1 Sistēmas arhitektūra

Sākumdati tiek ielādēti uz kopīgo tīkla datu glabātuvi, no kurienes pēc tam katrs serveris varēs pieprasīt savu daļu un pat laicīgi saglabāt pie sevis, lai netērēt laiku dārgām tīkla operācijām. Sākumdatu ielādes ietvaros var uzreiz centrālizēti un vienu reizi var uztaisīt visas vajadzīgas attēlu transformācijas un rezultātu saglabāt piemērotākā formātā.




*Attēls 9: Augstā līmeņa arhitektūra*

Piemēram attēlu samazināto versiju priekšsagatavošana kopējā vietā var samazināt datu pārraidi, paātrināt datu piegādi un ieekonomēt diska vietu apmācīšanas serveros. Kā arī izslēgt atkārtotas operācijas katrā epohā, paātrinot apmācīšanas procesu.




*Attēls 10: Asevišķa servera arhitektūra*

Šīs darbs apraksta datu ielādi viena servera ietvaros ar vienu videokārti.

## 2.2 Attēlu formāti

Attēlus var glabāt gan rastra, gan vektora formātā. Vektora formāti labāk piemēroti mazāk detalizēto attēlu definīcijai un nodrošinā ļoti kvalitātīvu attēla izmēra mērogošanas iespēju. Bet jo vairāk attēlam

detalizācijas, jo vairāk laika aizņem to ielāde un atspoguļošana, salīdzinot ar rastra attēliem. Tālāk tiks apskatīti tikai rastra formāti.

Palielinoties attēlu izšķirtspējai, rastra attēliem sāk augt arī datu izmērs un tas rada problēmu ar to glabāšanu un ielādes laiku. Līdz ar to tika ieviesta kompresija, kas samazina datu izmēru. Eksistē gan kompresija ar datu zaudējumiem, gan bez. Gandrīz vienmēr kompresija ar zaudējumiem samazina datu izmēru krietni labāk un pie tam ne vienmēr zaudējumus var redzēt, vai tie kaut ko ietekmē. Tā pati kompresija bez zaudējumiem tiek arī pielietota failu arhivēšanas formātos.

### 2.2.1 RAW

RAW ir attēla formāts, kas uzglabā tiešus datus no fotokameras sensora. Fotokameru sensoru pikseļi ir fotodiodes, kas tiek sadalītas uz sarkano, zaļo un zilo krāsām ar speciālu gaismas filtru palīdzību. Šīs fotodiodes pārveido gaismu elektriskajā signālā, kurš ir proporcionāls gaismas lielumam, kas nokrita uz to diodi. Šīs signāls tiek pastiprināts un pārveidots uz ciparu vērtību ar analogo-digitālo pārveidotāja (ADC) palīdzību.

Dažādu kameru sensoriem ir dažāda RGB sensoru izvietošana un tātad sarkano, zaļo un zilo fotodiožu izvietojums var atšķirties. Dažiem sensoriem ir Bayer filtru masīvs, kur katrs sensora pikselis ir vai nu sarkans, zaļš vai zils, kas sakārtots noteiktā shēmā. Citiem sensoriem, piemēram, Foveon X3, ir trīs slāņu sensors, kur katrs pikselis uztver visu sarkano, zaļo un zilo informāciju.

RAW attēla formāts saglabā neapstrādātos datus no attēla sensora, pirms tie tiek apstrādāti skatāmā attēlā. Šie neapstrādātie dati ietver ne tikai katrā pikselī uzņemtās gaismas intensitāti, bet arī informāciju par kameras sensoru un fotografēšanas apstākļiem, piemēram, baltā balansu, ISO un ekspozīcijas laiku. Pēc tam datus var apstrādāt un interpretēt ar programmatūru, lai izveidotu galīgo attēlu.

Rezumējot, RAW attēla formāts saglabā visus sākotnējos datus, kas uzņemti ar kameras attēla sensoru, tostarp informāciju par sensoru un fotografēšanas apstākļiem. Pēc tam attēla datus apstrādā programmatūra, lai izveidotu galīgo attēlu. Kameras ADC un pastiprinātājiem ir svarīga loma, pārveidojot sensora uztverto gaismu digitālā vērtībā, ko var saglabāt RAW attēlā.

ISO ir kameras gaismas jutības mērs. Jo augstāka ir ISO vērtība, jo jutīgāka ir kamera pret gaismu, tādējādi nodrošinot labāku veiktspēju vāja apgaismojuma apstākļos. Tomēr augstākas ISO vērtības var radīt arī lielāku attēla troksni, ko var uzskatīt par graudainību gala attēlā.

Pamata ISO ir zemākā ISO vērtība, ko kamera spēj izmantot. Šī vērtība parasti ir aptuveni 100, un tā tiek uzskatīta par kameras attēla sensora sākotnējo jutību. Pamata ISO režīmā kamera darbojas optimālā veiktspējas līmenī, ar viszemāko troksni un labāko attēla kvalitāti.

Fotografējot vāja apgaismojuma apstākļos, fotogrāfiem, iespējams, būs jāpalielina ISO vērtība, lai varētu uzņemt pareizi eksponētu attēlu. Palielinot ISO, kamera kļūst jutīgāka pret gaismu, nodrošinot ātrāku aizvara ātrumu vai plašāku diafragmas atvērumu, kas savukārt ļauj vairāk gaismas sasniegt sensoru. Tomēr, palielinoties ISO vērtībai, palielinās arī attēla trokšņa daudzums.

Tāpēc fotogrāfiem ir svarīgi ņemt vērā gan ISO vērtību, gan kompromisu starp attēla kvalitāti un troksni, izvēloties labāko ISO iestatījumu konkrētai fotografēšanas situācijai.

ISO palielināšana faktiski nemaina kameras sensora jutību. Tā vietā tas pavairo signālu no sensora, efektīvi pastiprinot signālu. Tādējādi tiek iegūts spilgtāks attēls, kas nodrošina labāku veiktspēju vāja apgaismojuma apstākļos.

Digitālajās kamerās ISO parasti tiek ieviests kā sensora signāla pastiprinājums pēc tam, kad ADC to ir pārveidojis par digitālo vērtību. Šī pastiprināšana tiek veikta, reizinot digitālo signālu ar pastiprinājuma koeficientu, kā rezultātā attēls kļūst gaišāks. Reizinot signālu, ISO vērtības palielināšana var palīdzēt uzlabot attēla kvalitāti vāja apgaismojuma apstākļos, taču tas arī palielina attēla trokšņa daudzumu. Iemesls tam ir tas, ka troksnis tiek pastiprināts kopā ar signālu, padarot to pamanāmāku galīgajā attēlā. Rezumējot, ISO palielināšana digitālajā kamerā faktiski nemaina sensora jutību, bet gan pastiprina signālu no sensora, tādējādi padarot kameru jutīgāku pret gaismu. Tomēr šis pastiprinājums arī palielina attēla trokšņa daudzumu. Dažiem kameru modeļiem ir vairāki pamata ISO, kas nozīmē, ka kamerai ir iespēja darboties dažādos īstos jutības līmeņos.

Vairāki pamata ISO nodrošina lielāku elastību fotogrāfiem, jo viņi var izvēlēties labāko pamata ISO konkrētajai fotografēšanas situācijai. Piemēram, fotografējot spilgtā dienasgaismā, viņi var izvēlēties zemāko pamata ISO – 100, kas nodrošina vislabāko attēla kvalitāti ar viszemāko trokšņu līmeni. Vāja apgaismojuma apstākļos var izvēlēties augstāku pamata ISO – 320, kas ļauj kamerai darboties ar lielāku gaismas jutību, nodrošinot lielāku aizvara ātrumu vai plašāku diafragmas atvērumu. Vairāku pamata ISO ir noderīga funkcija fotogrāfiem, jo tā ļauj maksimāli izmantot kameras iespējas un sasniegt vislabāko iespējamo attēla kvalitāti konkrētajā fotografēšanas situācijā.

Bayer filtru bloks ir krāsu filtra mozaīka, kas tiek novietota virs attēla sensora daudzās digitālajās kamerās. Filtrs sastāv no sarkaniem, zaļiem un ziliem filtriem, kuros ir divreiz vairāk zaļo filtru nekā sarkanā vai zilā krāsā. Šis modelis tiek izmantots, jo cilvēka acs ir visjutīgākā pret zaļo gaismu, un zaļie filtri ļauj precīzāk attēlot tveramo ainu. Katrs sensora pikselis ir pārklāts ar vienu krāsu filtru — sarkanu, zaļu vai zilu. Kad tiek uzņemts attēls, katrs pikselis reģistrē tikai tam atbilstošās krāsas gaismas intensitāti. Tas nozīmē, ka katram pikselim ir informācija tikai par vienu krāsu kanālu – sarkanu, zaļu vai zilu. Lai izveidotu pilnkrāsu attēlu, neapstrādātie attēla dati no Bayer filtru masīva tiek interpolēti, lai novērtētu trūkstošo krāsu informāciju katrā pikselī. Šis process, ko sauc par demosaicināšanu, izmanto sarežģītus algoritmus, lai novērtētu trūkstošās sarkanās un zilās vērtības, pamatojoties uz apkārtējiem pikseļiem. Pēc tam demosaicētais attēls tiek pārveidots par RGB attēlu, kas ir uzņemtās ainas pilnkrāsu attēlojums. Rezumējot, Bayer filtru masīvs ir krāsu filtra mozaīka, kas tiek novietota virs attēla sensora daudzās digitālajās kamerās. Filtru veido sarkanu, zaļu un zilu filtru raksts, un katrs sensora pikselis reģistrē tikai tai atbilstošās krāsas gaismas intensitāti. Pēc tam Bayer filtru masīva neapstrādātie attēla dati tiek interpolēti, lai novērtētu trūkstošo krāsu informāciju un izveidotu pilnkrāsu RGB attēlu.

### 2.2.2 BMP

BMP (Bitmap) faila formāts ir plaši izmantots failu formāts digitālo attēlu glabāšanai operētājsistēmās, kuru pamatā ir Windows. BMP ir vienkāršs un saprotams attēlu formāts, kas saglabā attēlus pa pikseļiem bez saspiešanas.

BMP faila formātu Microsoft ieviesa 1987. gadā kā patentētu formātu. Kopš tā laika to plaši izmanto dažādas operētājsistēmas, tostarp Windows, OS/2 un citas platformas. BMP faili ir nesaspiesti, kas nozīmē, ka tie var aizņemt ievērojamu vietu diskā. Tomēr tos ir ļoti vienkārši lasīt un rakstīt, padarot tos par ideālu formātu izstrādātājiem, kuriem programmatiski jāstrādā ar attēliem.

BMP faili sastāv no faila galvenes, bitkartes informācijas galvenes, krāsu tabulas un faktiskajiem bitkartes datiem. Faila galvenē ir informācija par failu, piemēram, tā lielums, veids un nobīde līdz bitkartes datu sākumam. Bitkartes informācijas galvenē ir informācija par pašu bitkarti, piemēram, tās platums, augstums, krāsu dziļums un saspiešanas metode (ja tāda ir). Krāsu tabula ir neobligāta sadaļa, kurā tiek glabāta attēla izmantotā krāsu palete, savukārt bitkartes dati satur faktiskos attēla datus pa pikseļiem.

BMP faili saglabā attēlus, izmantojot "pikseļu pa pikseļa" pieeju. Katrs pikselis ir attēlots ar vērtību kopu, kas nosaka tā krāsu. Pikseļa attēlošanai izmantoto vērtību skaits (t.i., krāsu dziļums) ir atkarīgs no izmantotā BMP formāta. Visizplatītākajā BMP formātā tiek izmantoti 24 biti uz pikseļu, kas nozīmē, ka katrs pikselis ir attēlots ar trīs vērtībām: vienu sarkanajam komponentam, vienu zaļajam komponentam un vienu zilajam komponentam.

BMP faili parasti nav saspiesti, kas nozīmē, ka tie var aizņemt ievērojamu vietu diskā. Tomēr ir pieejamas saspiešanas metodes, piemēram, Run-Length Encoding (RLE) saspiešanas metode. RLE saspiešana samazina bitkartes datu lielumu, kodējot pikseļus ar tādu pašu krāsu kā vienu vērtību.

Viena no galvenajām BMP formāta priekšrocībām ir tā vienkāršība. BMP failus ir ļoti viegli lasīt un rakstīt, padarot tos par ideālu formātu izstrādātājiem, kuriem programmatiski jāstrādā ar attēliem. Turklāt BMP failus atbalsta plašs lietojumprogrammu klāsts, tostarp attēlu redaktori, tīmekļa pārlūkprogrammas un operētājsistēmas. Tomēr BMP failiem ir arī vairāki trūkumi. Pirmkārt, tie parasti ir ļoti lieli, jo tie nav saspiesti. Tas var izraisīt lēnāku failu pārsūtīšanu un būtiskākas uzglabāšanas prasības. Turklāt BMP faili neatbalsta caurspīdīgumu vai alfa kanālus, padarot tos mazāk noderīgus noteiktām lietojumprogrammām. Visbeidzot, saspiešanas trūkums dažos gadījumos var izraisīt detaļu un krāsu precizitātes zudumu.

Atsauces:


- Microsoft. (2017). BMP file format. https://docs.microsoft.com/en-us/windows/win32/gdi/bitmap-file-format

### 2.2.3 PNG

Portatīvā tīkla grafika (PNG) ir populārs attēlu formāts, kas ir guvis plašu pielietojumu datorgrafikas pasaulē. PNG tika izveidots, lai pārvarētu dažus iepriekšējā GIF formāta ierobežojumus, tostarp patentu un licencēšanas problēmas, ierobežotu krāsu atbalstu un nespēju apstrādāt caurspīdīgumu. PNG ir atvērts bezatlīdzības formāts, kas atbalsta augstas kvalitātes attēlus ar pilnu krāsu diapazonu un dažādu caurspīdīguma pakāpi.

Pirmo reizi PNG 1995. gadā izveidoja izstrādātāju grupa, kas meklēja alternatīvu GIF formātam. GIF formāts piederēja uzņēmumam CompuServe, un tā lietošanai bija nepieciešama licence, kas ierobežoja tā pieņemšanu. PNG tika izstrādāts kā atvērts, brīvs no patentiem formāts, kas ļautu labāk saspiest un atbalstīt vairāk krāsu. Pirmā PNG versija PNG 1.0 tika izlaista 1996. gadā. Kopš tā laika ir veikti vairāki formāta atjauninājumi, tostarp PNG 1.1, PNG 1.2 un PNG 1.3. Jaunākā versija PNG 1.4 tika izlaista 2003. gadā, un tai tika pievienots atbalsts ICC profiliem un pelēktoņu ar alfa.

PNG ir bezzudumu saspiešanas formāts, kas nozīmē, ka saspiešanas un dekompresijas laikā tiek saglabāti sākotnējie attēla dati. Tas ir pretstatā zudumiem saspiešanas formātiem, piemēram, JPEG, kas izmet dažus sākotnējos attēla datus, lai iegūtu mazāku failu izmēru. PNG izmanto saspiešanas paņēmienu kombināciju, tostarp DEFLATE algoritmu, kas tiek izmantots arī ZIP faila formātā. PNG atbalsta arī alfa caurspīdīgumu, kas ļauj attēlot attēlus ar dažādu caurspīdīguma pakāpi jebkurā fona krāsā vai attēlā. Tas ir noderīgi, lai izveidotu grafiku ar mīkstām malām, ēnām un citiem efektiem. PNG atbalsta arī pīšanos, kas ļauj pakāpeniski parādīt attēlu, kad tas tiek lejupielādēts no vietnes vai cita avota. Tas var būt īpaši noderīgi lieliem attēliem, kuru ielāde pretējā gadījumā prasītu ilgu laiku.

PNG tiek plaši izmantots datorgrafikā, īpaši attēliem, kuriem nepieciešama caurspīdīgums vai pilnkrāsu atbalsts. PNG bieži izmanto logotipiem, ikonām un citai grafikai, kas jāparāda uz dažādiem foniem. PNG ir populāra arī tīmekļa dizainā, kur to bieži izmanto kopā ar Cascading Style Sheets (CSS), lai izveidotu vizuāli pievilcīgas un atsaucīgas vietnes. Turklāt PNG atbalsta lielākā daļa mūsdienu tīmekļa pārlūkprogrammu un attēlu rediģēšanas programmatūras, padarot to par daudzpusīgu un plaši izmantotu attēlu formātu.

Atsauces:


1. "Portable Network Graphics (PNG) Specification (Second Edition)" (2003). W3C Recommendation. Retrieved from https://www.w3.org/TR/PNG/.
2. Boutell, T. (1997). "The PNG (Portable Network Graphics) Home Site". https://www.libpng.org/pub/png/.
3. Roelofs, G. (1999). "PNG: The Definitive Guide". O'Reilly Media. http://www.libpng.org/pub/png/pngbook.html

### 2.2.4 TIFF

Tagged Image File Format (TIFF) ir plaši izmantots attēla formāts, kas ir pazīstams ar savu elastību, pārnesamību un spēju saglabāt augstas kvalitātes attēlus. Šī tehniskā apraksta mērķis ir sniegt visaptverošu skaidrojumu par TIFF formātu, tostarp tā struktūru, kodējumu un lietojumiem.

TIFF faili sastāv no galvenes, kam seko viens vai vairāki attēlu failu direktoriji (IFD). Galvenē ir ietverta pamatinformācija par TIFF failu, piemēram, tā baitu secība un pirmā IFD atrašanās vieta. Katrs IFD satur tagu kopu, kas sniedz detalizētu informāciju par attēla datiem, piemēram, tā platumu, augstumu, krāsu dziļumu un saspiešanas veidu. TIFF faili var ietvert arī papildu IFD vairāku attēlu, attēlu sīktēlu vai metadatu glabāšanai.

TIFF failus var kodēt, izmantojot dažādus saspiešanas algoritmus, piemēram, LZW, JPEG vai PackBits. LZW kompresija tiek plaši izmantota, jo tā nodrošina labus saspiešanas koeficientus, būtiski nezaudējot attēla kvalitāti. Tomēr dažās valstīs tas var būt saistīts ar patentiem. JPEG saspiešana ir izplatīta arī fotogrāfisku attēlu glabāšanai, taču tā var radīt saspiešanas artefaktus un nav piemērota attēlu ar asām malām vai teksta glabāšanai. PackBits saspiešana ir vienkāršs izpildes garuma kodēšanas algoritms, kas ir efektīvs nelielu identisku pikseļu sēriju saspiešanai.

TIFF faili tiek plaši izmantoti daudzās lietojumprogrammās, kurās nepieciešami augstas kvalitātes attēli, piemēram, digitālajā fotogrāfijā, drukāšanā un dokumentu skenēšanā. Tos izmanto arī medicīnisko attēlu, zinātnisko datu un ĢIS datu glabāšanai. TIFF faili var saglabāt attēlus dažādās krāsu telpās, tostarp RGB, CMYK, pelēktoņu un indeksētās krāsās. Tie var arī saglabāt caurspīdīguma informāciju, vairākus alfa kanālus un ICC profilus krāsu pārvaldībai.

TIFF formāts ir elastīgs un plaši izmantots attēlu formāts, kas nodrošina bagātīgu funkciju kopumu augstas kvalitātes attēlu glabāšanai. Tās struktūra, kodējums un lietojumi ir sīki aprakstīti šajā tehniskajā aprakstā. Papildinformāciju par TIFF formātu var atrast Adobe Systems Incorporated publicētajā TIFF specifikācijas dokumentā un Aldus Corporation publicētajā TIFF 6.0 specifikācijā.

Atsauces:

1. Adobe Systems Incorporated. (1992). TIFF (Tag Image File Format) Specification. https://www.adobe.io/content/dam/udp/en/open/standards/tiff/TIFF6.pdf

### 2.2.5 JPEG

Digitālajā laikmetā attēlus izmanto dažādās lietojumprogrammās, sākot no sociālajiem medijiem līdz zinātniskiem pētījumiem. Tomēr augstas kvalitātes attēlu uzglabāšana un pārsūtīšana var būt sarežģīta to lielā izmēra dēļ. Lai risinātu šo problēmu, ir izstrādātas dažādas attēlu saspiešanas metodes, no kurām viena ir JPEG formāts. JPEG ir zudumu saspiešanas metode, kas samazina digitālo attēlu izmēru, vienlaikus saglabājot to kvalitāti līdz noteiktai pakāpei.

JPEG saspiešanas process sastāv no diviem posmiem: diskrētās kosinusa transformācijas (DCT) un kvantēšanas. DCT posmā attēls tiek sadalīts 8x8 pikseļu blokos, un katrs bloks tiek pārveidots savā frekvenču domēnā, izmantojot DCT algoritmu. Rezultāts ir 64 frekvenču koeficientu kopa, kas attēlo sākotnējo bloku. DCT posms samazina attēla attēlošanai nepieciešamās informācijas apjomu, jo lielākā daļa augstfrekvences komponentu parasti ir mazi un tos var izmest.

Pēc DCT stadijas tiek piemērots kvantēšanas posms. Kvantēšana ietver katra frekvences koeficienta dalīšanu ar kvantēšanas matricu, kuras pamatā ir vēlamais saspiešanas līmenis. Tā rezultātā tiek zaudēta daļa informācijas, jo kvantēšanas matricas dēļ daži koeficienti tiek noapaļoti līdz nullei. Tomēr zuduma pakāpi var kontrolēt, pielāgojot kvantēšanas matricu. Augstāks kompresijas līmenis rada lielāku zudumu pakāpi.

Pēc tam kvantētie koeficienti tiek tālāk saspiesti, izmantojot mainīga garuma kodēšanu (VLC). Šis kodēšanas paņēmiens piešķir īsākus kodus bieži sastopamiem simboliem un garākus kodus retāk sastopamiem simboliem. Rezultāts ir saspiests attēls, ko var efektīvi pārsūtīt vai saglabāt.

JPEG formāta priekšrocības un trūkumi: JPEG formātam ir vairākas priekšrocības, tostarp augsta saspiešanas pakāpe un savietojamība ar plašu programmatūras un aparatūras platformu klāstu. Tā atbalsta arī progresīvo saspiešanu, kas ļauj pārsūtīt un parādīt attēlus pakāpeniski, sākot ar zemas kvalitātes versiju, kas tiek pakāpeniski uzlabota, līdz tiek sasniegta pilna attēla kvalitāte. JPEG ir piemērots arī fotogrāfiju attēlu saspiešanai, jo var saglabāt smalkas detaļas un faktūras.

Tomēr JPEG saspiešana ir ar zaudējumiem, kas nozīmē, ka saspiešanas procesā neizbēgami tiek zaudēta daļa attēla kvalitātes. Turklāt saspiešana var radīt vizuālus artefaktus, piemēram, bloķēšanu un izplūšanu, īpaši ar augstu saspiešanas pakāpi. Tas var ierobežot JPEG izmantošanu lietojumprogrammām, kurām nepieciešami augstas kvalitātes attēli, piemēram, medicīniskā attēlveidošana un zinātniskie pētījumi. JPEG formāta lietojumprogrammas: JPEG tiek plaši izmantots dažādās lietojumprogrammās, tostarp digitālajā fotogrāfijā, tīmekļa dizainā un multivides satura veidošanā. To izmanto arī satelītattēlu veidošanā, attālajā uzraudzībā un medicīniskajā attēlveidošanā, kur nepieciešami augstas kvalitātes attēli, bet lielu attēlu failu glabāšana un pārsūtīšana ir nepraktiska. Turklāt JPEG tiek izmantots video saspiešanā, kur katrs video kadrs tiek saspiests, izmantojot JPEG formātu.

Atsauces:


1. "JPEG image compression using discrete cosine transform" by S. Arivazhagan and L. Kannan (International Journal of Computer Science and Information Technologies, 2011) Link: https://arxiv.org/pdf/1405.6147.pdf
2. Image Classification in JPEG Compression Domain for Malaria Infection Detection https://www.researchgate.net/publication/360349350_Image_Classification_in_JPEG_Compression_Domain_for_Malaria_Infection_Detection
3. "The JPEG still picture compression standard" by W. B. Pennebaker and J. L. Mitchell https://ieeexplore.ieee.org/document/125072

### 2.2.6 WebP

WebP ir Google 2010. gadā izstrādāts attēla formāts, kas paredzēts, lai nodrošinātu uzlabotu attēla saspiešanu, vienlaikus saglabājot attēla kvalitāti. Formāts izmanto saspiešanas ar zudumiem un bezzudumu paņēmienu kombināciju, kā rezultātā faila izmēri ir mazāki, salīdzinot ar citiem populāriem attēlu formātiem, piemēram, JPEG un PNG. Šajā rakstā mēs sniegsim WebP formāta tehnisko aprakstu un skaidrojumu, tostarp tā galvenās funkcijas, saspiešanas algoritmus un priekšrocības. Mēs arī apspriedīsim tā saderību ar dažādām tīmekļa pārlūkprogrammām un salīdzināsim tā veiktspēju ar citiem attēlu formātiem. Mūsu analīze ir balstīta uz zinātnisko avotu pārskatu un publicētajiem WebP pētījumiem.

Attēli ir būtiska tīmekļa sastāvdaļa un tiek plaši izmantoti dažādiem mērķiem, piemēram, zīmola veidošanai, lietotāju iesaistīšanai un informācijas parādīšanai. Tomēr attēli var būtiski ietekmēt vietnes veiktspēju, īpaši, ja tie ir lieli. Tas var izraisīt lēnāku lapas ielādes laiku, kas var negatīvi ietekmēt lietotāja pieredzi. Lai risinātu šo problēmu, Google izstrādāja WebP formātu, kura mērķis ir nodrošināt līdzsvaru starp attēla kvalitāti un faila lielumu.

WebP ir atvērtā pirmkoda attēla formāts, kas izmanto gan zudumu, gan bezzudumu saspiešanas algoritmus, lai sasniegtu mazākus failu izmērus. Formāts ir balstīts uz VP8 video kodeku, ko arī izstrādāja Google. WebP izmanto divas galvenās saspiešanas metodes, proti, paredzamo kodēšanu un transformācijas kodēšanu.

Prognozējošā kodēšana ietver katra attēla pikseļa krāsas prognozēšanu, pamatojoties uz tuvumā esošo pikseļu vērtībām. Tas samazina attēla saglabāšanai nepieciešamo datu apjomu. Transformācijas kodēšana ietver pikseļu vērtību pārveidošanu jaunā attēlojumā, ko var efektīvāk saspiest. Šo paņēmienu izmanto kompresijā ar zaudējumiem, kad daļa informācijas tiek izmesta, lai iegūtu mazāku failu izmēru.

WebP ietver arī funkciju, ko sauc par alfa kanālu saspiešanu, ko izmanto caurspīdīgu pikseļu saspiešanai. Šī funkcija izmanto paņēmienu, ko sauc par alfa plaknes kodēšanu, kas identificē attēla apgabalus ar zemu caurspīdīgumu un saspiež tos agresīvāk.

WebP atbalsta vairākas tīmekļa pārlūkprogrammas, tostarp Google Chrome, Microsoft Edge un Opera. Tomēr Safari un Internet Explorer to neatbalsta. Lai nodrošinātu saderību, WebP ir iekļauts arī rezerves mehānisms, kas parāda attēla JPEG vai PNG versiju, ja pārlūkprogramma neatbalsta WebP.

WebP ir izstrādāts, lai nodrošinātu labāku saspiešanu salīdzinājumā ar citiem populāriem attēlu formātiem. Saskaņā ar Google veikto pētījumu, WebP var sasniegt līdz pat 34% mazāku failu izmēru salīdzinājumā ar JPEG un līdz 26% mazāku failu izmēru salīdzinājumā ar PNG, vienlaikus saglabājot līdzīgu attēla kvalitāti.

Atsauces:


1. WebP Specification: https://developers.google.com/speed/webp/docs/riff_container
2. WebP Image Format: https://www.w3.org/TR/WEBP/
3. WebP Metadata Specification: https://developers.google.com/speed/webp/docs/webp_metadata

4. WebP Lossless Bitstream Specification: https://developers.google.com/speed/webp/docs/webp_lossless_bitstream_specification

### 2.2.7 Formātu salīdzinājums

1. Diska vietas patēriņš:

- JPEG faili parasti ir mazāki, salīdzinot ar citiem attēlu formātiem, īpaši, ja ir augsta saspiešana. Tomēr precīzs izmērs ir atkarīgs no saspiešanas līmeņa, izšķirtspējas un attēla kvalitātes.
- TIFF faili parasti ir lielāki nekā JPEG faili, jo tie ir bezzudumu saspiešanas vai vispār nav saspiesti. Izmērs var atšķirties atkarībā no krāsas dziļuma un izšķirtspējas.
- PNG faili ir lielāki par JPEG failiem, bet mazāki par TIFF failiem. Tie izmanto bezzudumu saspiešanu un var arī saglabāt caurspīdīguma informāciju, kas palielina faila lielumu.
- WebP faili parasti ir mazāki par JPEG un PNG failiem. Tie izmanto saspiešanas metodes ar zaudējumiem un bezzudumu, kā arī var saglabāt caurspīdīguma informāciju.
- BMP faili parasti ir lielāki nekā JPEG, PNG un WebP faili. Tie neizmanto saspiešanu vispār, kas noved pie lielākiem faila izmēriem.

2. CPU slodze:

- JPEG failus var ātri un viegli atšifrēt. Tomēr JPEG failu saspiešana var būt procesora intensīva, it īpaši, ja tiek izmantots augsts saspiešanas līmenis.
- TIFF failus var ātri atšifrēt, taču to kodēšana var būt procesora intensīva, jo tie tiek saspiesti bez zudumiem.
- PNG failus var ātri atšifrēt un kodēt, taču kodēšanai var būt nepieciešams procesors, ja tiek izmantots lielāks krāsu dziļums vai lielāks attēla izmērs.
- WebP failus var ātri un efektīvi atšifrēt, taču to saspiešana var būt procesora intensīva, it īpaši, ja tiek izmantota bezzudumu saspiešanas metode.
- BMP failus var ātri kodēt un atšifrēt, jo tie neizmanto nekādu saspiešanu.

3. RAM pateriņš:

- JPEG faili parasti izmanto mazāk RAM nekā citi formāti dekodēšanas laikā, tāpēc tie ir laba izvēle attēlu parādīšanai tīmekļa lapās.
- TIFF faili var izmantot daudz RAM, īpaši strādājot ar lieliem attēliem vai lielu krāsu dziļumu.
- PNG faili var izmantot vairāk RAM nekā JPEG faili to saspiešanas algoritma dēļ.
- WebP faili var izmantot vairāk RAM nekā JPEG faili to saspiešanas algoritma dēļ.
- BMP faili var izmantot vairāk RAM nekā citi formāti, īpaši strādājot ar lieliem attēliem vai lielu krāsu dziļumu.

4. Krāsu bitu dziļums:

- JPEG faili atbalsta līdz 24 bitu krāsu dziļumu.

- TIFF faili atbalsta līdz 48 bitu krāsu dziļumu RGB attēliem un 32 bitu krāsu dziļumu CMYK attēliem.
- PNG faili atbalsta līdz 48 bitu krāsu dziļumu RGB attēliem un 32 bitu krāsu dziļumu pelēktoņu attēliem.
- WebP faili atbalsta līdz 24 bitu krāsu dziļumu kompresijai ar zaudējumiem un līdz 32 bitu krāsu dziļumu bezzudumu saspiešanai.
- BMP faili atbalsta līdz 32 bitu krāsu dziļumu RGB attēliem un 24 bitu krāsu dziļumu pelēktoņu attēliem.

5. Citi faktori:

- JPEG faili tiek plaši izmantoti tīmekļa grafikā un fotogrāfijās to mazā izmēra un labās attēla kvalitātes dēļ.
- TIFF faili parasti tiek izmantoti profesionālai fotografēšanai un drukāšanai, jo tie ir bezzudumu saspiešanas un atbalsta augstu krāsu dziļumu.
- PNG faili bieži tiek izmantoti tīmekļa grafikai un attēliem, kuriem nepieciešama caurspīdīgums.
- WebP faili kļūst arvien populārāki tīmekļa grafikai un attēliem to mazā izmēra un atbalsta caurspīdīguma un animācijas dēļ.
- BMP faili netiek plaši izmantoti to lielā izmēra un saspiešanas trūkuma dēļ.

Strādājot ar mašīnmācīšanos, izmantojot PyTorch un Inception modeļus, attēla formāta izvēle var ietekmēt apmācības procesu un iegūtā modeļa veiktspēju. Šeit ir daži apsvērumi:

1. Attēlu priekšapstrāde: pirms attēlu ievadīšanas mašīnmācīšanās modelī tie parasti ir iepriekš jāapstrādā, lai normalizētu pikseļu vērtības un pielāgotu izmēru un formu. Iepriekšējās apstrādes darbības var atšķirties atkarībā no modeļa un ievades formāta. Dažiem attēlu formātiem, piemēram, PNG un JPEG, var būt nepieciešama mazāka pirmapstrāde salīdzinājumā ar citiem, piemēram, TIFF un BMP.

2. Diska patēriņš: apmācības procesa laikā ievades attēli parasti tiek ielādēti atmiņā un padeves modelim pa partijām. Diska patēriņš var ietekmēt datu ielādes procesa ātrumu un efektivitāti. Kā minēts iepriekš, JPEG un WebP failiem parasti ir mazāki failu izmēri salīdzinājumā ar citiem formātiem, kas var paātrināt to ielādi no diska.

3. CPU slodze un RAM patēriņš: attēla formāts var ietekmēt arī CPU un RAM patēriņu apmācības procesā, jo īpaši datu ielādes un palielināšanas darbību laikā. Dažiem attēlu formātiem var būt nepieciešams vairāk CPU un RAM resursu nekā citiem to saspiešanas algoritma vai krāsu dziļuma dēļ. Piemēram, TIFF failiem ar lielu krāsu dziļumu var būt nepieciešams vairāk CPU un RAM resursu, salīdzinot ar JPEG failiem.

4. Modeļa veiktspēja: attēla formāta izvēle var ietekmēt arī mašīnmācīšanās modeļa veiktspēju. Daži attēlu formāti var radīt artefaktus vai trokšņus, kas var ietekmēt modeļa precizitāti. Piemēram, JPEG faili ar augstu saspiešanas līmeni var radīt saspiešanas artefaktus, kas var ietekmēt modeļa spēju atšķirt funkcijas. Parasti mašīnmācīšanās uzdevumos priekšroka tiek

dota bezzudumu saspiešanas formātiem, piemēram, PNG un WebP, jo tie saglabā attēla kvalitāti, neieviešot artefaktus.

Strādājot ar PyTorch un Inception modeļiem, attēla formāta izvēle ir atkarīga no vairākiem faktoriem, tostarp attēla priekšapstrādes, diska patēriņa, CPU un RAM patēriņa un modeļa veiktspējas. Parasti mašīnmācīšanās uzdevumos var dot priekšroku bezzudumu saspiešanas formātiem, piemēram, PNG un WebP, jo tie saglabā attēla kvalitāti, neieviešot artefaktus, savukārt JPEG var būt laba izvēle attēliem ar mazāku sarežģītību un mazāku failu izmēru.

Salīdzinājums uz 9504x6336 pikseļu attēla piemēra. Visas versijas tika saglabātas iekš Adobe Photoshop v23.4.1, konvertējot no oriģināla.

| Formāts | Biti | Kvalitāte / kompressija | Izmērs |
|---|---|---|---|
| BMP | 8 | 100% / nav | 172 MB |
| PNG | 8 | 100% / maksimums | 53.1 MB |
| PNG | 16 | 100% / maksimums | 241 MB |
| TIFF | 8 | 100% / nav | 172 MB |
| TIFF | 8 | 100% / lzw | 55 MB |
| TIFF | 8 | 100% / zip | 52 MB |
| TIFF | 16 | 100% / nav | 344 MB |
| TIFF | 16 | 100% / lzw | 353 MB |
| TIFF | 16 | 100% / zip | 282 MB |
| JPEG | 8 | maksimums | 19.5 MB |
| JPEG | 8 | minimums | 1.01 MB |
| WebP | 8 | 100% lossless | 37.9 MB |
| WebP | 8 | maksimums lossy | 2.03 MB |
| WebP | 8 | minimums lossy | 236 KB |

## 2.3 Attēlu krāsu telpas

Attēla krāsu telpas ir matemātiski modeļi, kas definē krāsu diapazonu, ko var attēlot digitālajos attēlos. Tās nodrošina standartizētu veidu, kā aprakstīt krāsas tā, lai tās varētu precīzi attēlot dažādos ierīcēs, piemēram, datoru monitoros un printeros.

sRGB (standarta Sarkanais Zaļais Zils) ir plaši izmantota krāsu telpa digitālajiem attēliem. Tā balstās uz RGB modeli un definē standarta krāsu gamma, kas optimizēta tipiskiem datoru monitoriem un web lietojumiem.

AdobeRGB ir vēl viena RGB bāzēta krāsu telpa, kas piedāvā plašāku krāsu diapazonu nekā sRGB. To bieži izmanto profesionālās fotogrāfijas un drukas lietojumos.

YUV ir krāsu telpa, kas atdala krāsu informāciju no spilgtuma informācijas attēlā. Tas padara to īpaši noderīgu lietojumiem, kas prasa atsevišķu krāsu un spilgtuma informācijas apstrādi, piemēram, video saspiešanai un kodēšanai.

Luv un Lab ir divas krāsu telpas, kas ir izstrādātas, lai būtu uztveres vienmērīgas, kas nozīmē, ka vienādi attālumi krāsu telpā atbilst vienādām uztveres atšķirībām krāsās. Tos bieži izmanto lietojumos, piemēram, krāsu korekcijā un kalibrēšanā un krāsu uztveres pētījumos.

XYZ ir krāsu telpa, kas nodrošina standarta, ierīču neatkarīgu veidu, kā aprakstīt krāsas. Tā definē trīs primārkrāsas, kas tiek izmantotas, lai attēlotu visas krāsas redzamajā spektrā.

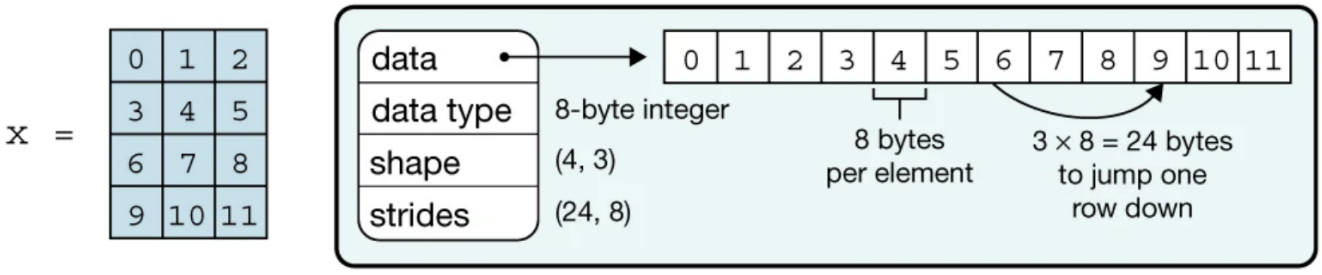
ProPhoto ir krāsu telpa, kas piedāvā ļoti plašu krāsu diapazonu, kas padara to noderīgu lietojumiem, piemēram, augstas kvalitātes drukāšanai un profesionālajai fotogrāfijai. Tas balstās uz RGB modeli un ietver krāsas, kas atrodas ārpus vairuma citu krāsu telpu diapazona.

Kopumā krāsu telpas nodrošina standartizētu veidu, kā attēlot krāsas digitālajos attēlos. Dažādām krāsu telpām ir dažādi priekšrocības un mīnusi, un tās tiek izmantotas atkarībā no konkrētā lietojuma un attēla prasībām. Izmantojot pareizo krāsu telpu, var nodrošināt precīzākas un uztveres vienmērīgākas krāsu atveides digitālajos attēlos, tādējādi uzlabojot to kvalitāti un atbilstību oriģinālajiem attēliem.

## 2.4 Datu glabāšanas formāti

### 2.4.1 NumPy masīvs

NumPy masīvs ir datu struktūra, kas efektīvi glabā daudzdimensionālus masīvus (tenzorus) operatīvājā atmiņā un ļauj efektīvi apstrādāt tos ar centrālprocesora (CPU) palīdzību. NumPy masīvs iekļauj sevī norādi uz atmiņu, kur glabājas dati, kā arī datu tipu, datu "formu" un "soļus".




*Attēls 11: NumPy masīva iekšeja struktūra*

### 2.4.2 HDF5

HDF5 formātu var uzskatīt par failu sistēmu, kas ietverta un aprakstīta vienā failā. Vienā HDF5 failā varat saglabāt līdzīgu datu kopu, kas sakārtota tādā pašā veidā, kā faili un mapes sakārtoti datorā. Tomēr HDF5 failā "direktorijias", sauc par grupām, un “failus” sauc par datu kopām.

- **Grupa:** Direktorijas tipa elements iekš HDF5 faila kas var ietvērt sevī citas grupas vai datu kopas.
- **Datu kopa:** Faktiskie dati, kas ietverti HDF5 failā. Datu kopas bieži tiek glabātas (nav obligāti) grupās.

Hierarhiskais datu formats tiek domāts liela apjoma datu glabāšanai un organizēšanai. Hierarhijas tiek definētas POSIX-veida sintakša veidā, piemēram “/path/to/dataset”.

### 2.4.3 Zarr

Zarr ir moderna alternatīva NumPy-veidīgo N-dimensionālo masīvu glabāšanai atmiņā, uz diska, ZIP arhīvā, AWS S3 glabātuvē vai atslēg-vērtību datubāzē. Zarr funkcionālitāti var arī paplašināt ar citu glabātuvi. Tā atbalsta paralēlo lasīšanu un rakstīšanu no dažādiem procesiem. Datu masīvus var organizēt hierarhijās. Tāpat kā HDF5 ir iespēja definēt datu gabala izmēru, ka arī kompresēt datu gabalus, pielagojot to efektīvakai lasīšanai. Šī metode ir noderīga datu lasīšanai no tīkla.

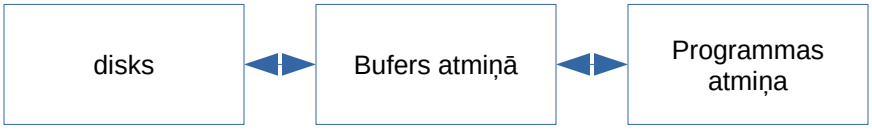
## 2.5 Datu ielādes metodes

### 2.5.1 Failu metode

Šī ir visvienkāršā metode. Notiek tikai lejupielādēto datu saglabāšana failos uz cietā diska failu sistēmas, no kuras pēc tam notiks ielāde. Bildes tiks iepriekš pārveidotas (samazinātas, apgrieztas) lai netērētu laiku un resursus transformācijai katrā epohā no jauna.

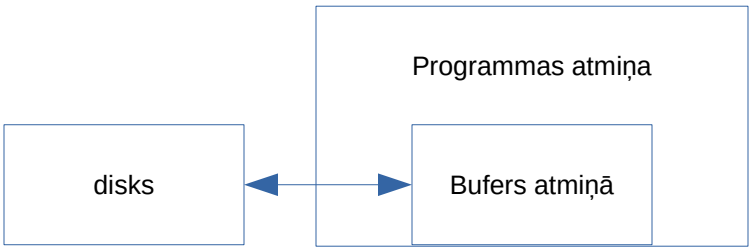
### 2.5.2 Numpy mmap

Mmap (memory map) funkcionālitāte ļauj programatūrai interpretēt saturu uz diska kā saturu operātīvajā atmiņā. Bez **mmap()** faila ielāde notiek šādi:




*Attēls 12: Parastais datu ielādes princips*

Kad programma lasa faila saturu, operētājsistēma vispirms ielāde to saturu buferā pa daļām (lapām) un pēc tam programmas atmiņā. Ja fails ir lielāki par brīvas operatīvās atmiņas apjomu, tad operētājsistēma sāks izspiest datus uz mijmaiņas failu, ja tāds vispār ir. Tas krietni palēninās visas sistēmas ātrdarbību. Ja mijmaiņas faila nav, vai pat tājā vieta izbeidzās, tad apstās kādu programmu, kura mēģinās tājā brīdī pieprāsīt vairāk atmiņas.



*Attēls 13: mmap() darbības princips*

**Mmap()** gadījumā buferis tiek iebūvēts programmas atmiņā, izslēdzot lieko datu pārraides soli.

Metodei ir arī trūkumi:

- Datiem jāatrodas uz cietā diska lokāli, tie nevar atrasties attālinātos resursos. (Tomēr ir daži risinājumi, lai pieslēgtu attālinātus resursus kā lokālus)
- N-dimensionālo masīvu griezumi var krietni palielināt lasījumu skaitu no diska.

### 2.5.3 CuPy mmap

CuPy ir atvērta koda bibliotēka darbam ar masīviem priekš GPU-paātrinātiem skaitļošanas uzdevumiem ar Python programmēšanas valodas palīdzību un ir līdzīga NumPy. CuPy izmanto CUDA rīku komplekta bibliotēkas, tādas kā cuBLAS, cuRAND, cuSOLVER, cuSPARSE, cuFFT, cuDNN un NCCL lai pilnībā izmantotu GPU arhitektūru.[6] Tā ielādē datu masīvu uz VRAM un tiek paziņots, ka PyTorch var izmantot to bez liekas datu pārraides caur DLPack[7].

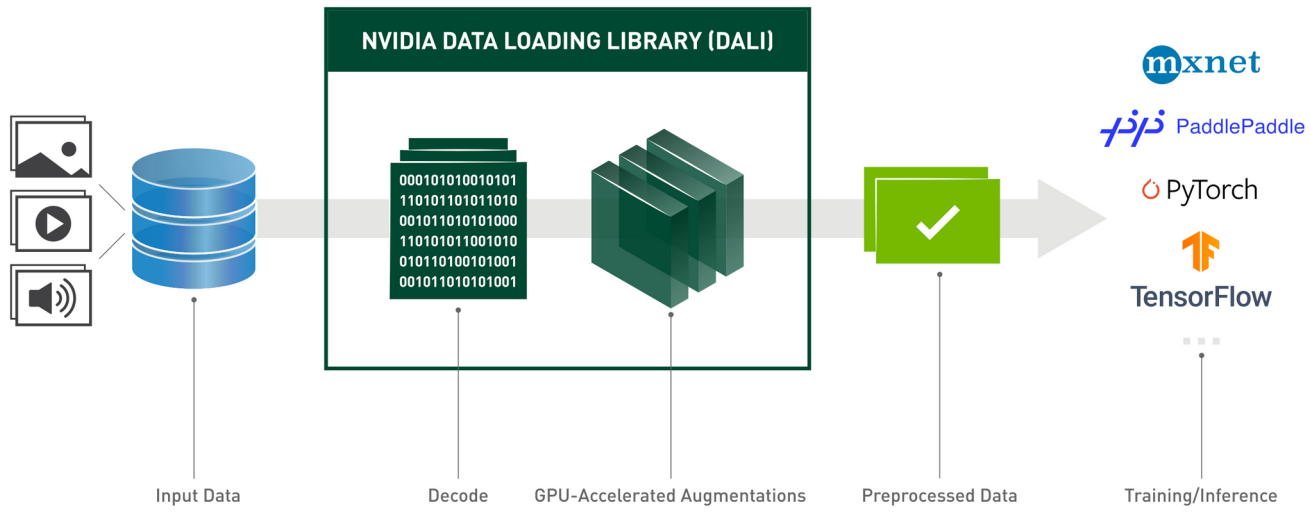
### 2.5.4 Nvidia DALI + Nvidia GPUDirect Storage

The NVIDIA Data Loading Library (DALI) is a library for data loading and pre-processing to accelerate deep learning applications. It provides a collection of highly optimized building blocks for loading and processing image, video and audio data. It can be used as a portable drop-in replacement for built in data loaders and data iterators in popular deep learning frameworks.

Deep learning applications require complex, multi-stage data processing pipelines that include loading, decoding, cropping, resizing, and many other augmentations. These data processing pipelines, which are currently executed on the CPU, have become a bottleneck, limiting the performance and scalability of training and inference.

DALI addresses the problem of the CPU bottleneck by offloading data preprocessing to the GPU. Additionally, DALI relies on its own execution engine, built to maximize the throughput of the input pipeline. Features such as prefetching, parallel execution, and batch processing are handled transparently for the user.

In addition, the deep learning frameworks have multiple data pre-processing implementations, resulting in challenges such as portability of training and inference workflows, and code maintainability. Data processing pipelines implemented using DALI are portable because they can easily be retargeted to TensorFlow, PyTorch, MXNet and PaddlePaddle.



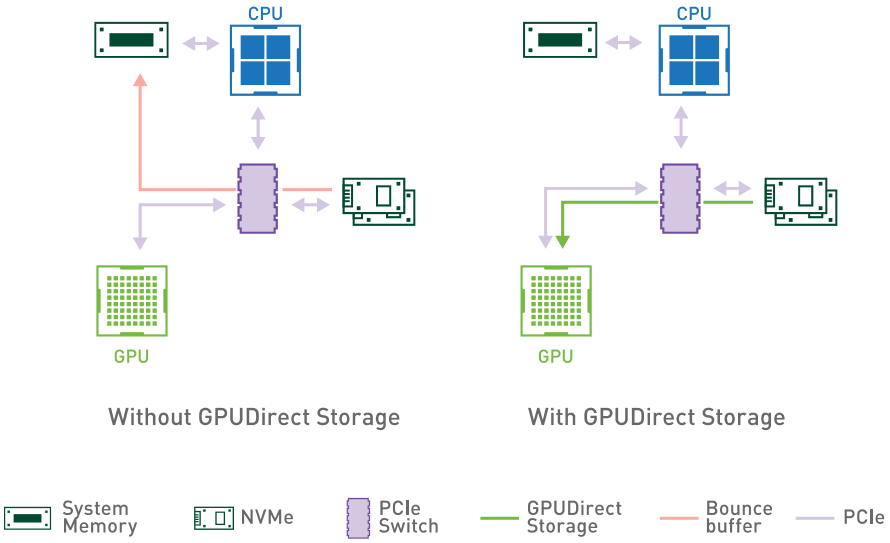

*Attēls 14: Nvidia DALI pielietojuma shēma*

Nvidia DALI īpatnības:

- Viegli izmantojama funkcionāla stila Python API.
- Dažādu datu formātu atbalsts - LMDB, RecordIO, TFRecord, COCO, **JPEG**, JPEG 2000, WAV, FLAC, OGG, H.264, VP9 un HEVC.
- Pārnesamā starp populārām dziļas mācīšanas bibliotēkām: TensorFlow, **PyTorch**, MXNet, PaddlePaddle.
- Atbalsta operāciju izpildi gan centrālprocesorā (CPU), gan videokartes procesorā (GPU).
- Scalable across multiple GPUs.
- Flexible graphs let developers create custom pipelines.
- Extensible for user-specific needs with custom operators.
- Accelerates image classification (ResNet-50), object detection (SSD) workloads as well as ASR models (Jasper, RNN-T).
- Atbalsa taisnu datu pārraidi starp krātuvi un videokartes atmiņu ar GPUDirect Storage palīdzību.
- Easy integration with NVIDIA Triton Inference Server with DALI TRITON Backend.
- Atvērts kods

Galvenā problēma visām iepriekšejām metodēm ir tas, ka dati tiek ielādēti no diska caur CPU uz RAM, dažreiz ar dažādu buferu palīdzību notiek “lieka” datu kopēšana RAM ietvaros un tikai tad dati tiek

pārvietoti uz videoatmiņu. Nvidia piedavā savu datu ielādes mehānismu caur CUDA funkcionālitāti, kas ļauj ielādēt datus no diska pa tiešo uz videoatmiņu. Galvenais nosacījums ir: Operētājsistēmai ir jābūt Linux 5.4, videokārtei no Nvidia ar CUDA 10.1+ atbalstu, un gan diskam, gan videokartei ir jāstrāda caur PCI Express kopni[8].




*Attēls 15: Datu ielādes principu salīdzinājums*

Ir līdzīga funkcionalitāte no Microsoft – DirectStorage API[9], kas cenšas paveikt to pašu pie nosacījuma, ka programmatūra strādā uz Windows 11 ar Direct3D un iekārtas tiek pieslēgtas PCI Express kopnei. Kāmēr citas metodes vairāk risina datu izmēra un glabāšanas problēmas, DirectStorage rīki mēģina inovatīvi atrisināt tieši datu ielādes ātruma problēmas. Lai veiksmīgi to paveiktu, datiem jābūt iepriekš sagatavotiem darbam ar videokarti, lai nevajadzētu iesaistīt centrālprocesoru.

Nvidia datu ielādes bibliotēka (DALI) ir pārnēsājama, atvērtā koda bibliotēka attēlu, video un runas dekodēšanai un papildināšanai, lai paātrinātu dziļas mācīšanās lietojumprogrammas. DALI samazina latentumu un apmācības laiku, mazinot vājās vietas, pārklājot apmācību un iepriekšēju apstrādi. Tas nodrošina iebūvēto datu ielādētāju un datu iteratoru nomaiņu populārajās dziļās apmācības ietvaros, lai tos varētu viegli integrēt vai atkārtoti atlasīt dažādās sistēmās[10].

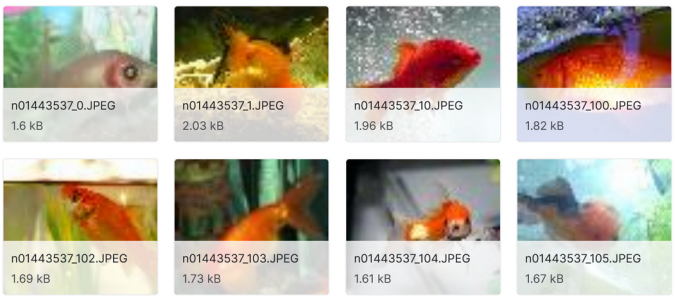
## 2.6 Datu kopas

### 2.6.1 CIFAR10

*Attēls 16: CIFAR-10 datu kopnes attēlu piemērs*

CIFAR-10 datukopa satur 60 tūsktošus 32x32 krāsainas bildes, kuras ir sadalītas 10 klasēs, katrai klasei atbils 6000 bildes. Datukopa ir sadalīta uz 50000 bildēm apmācībai un 10000 bildēm pārbaudei.

### 2.6.2 Tiny ImageNet




*Attēls 17: Tiny ImageNet datu kopnes attēlu piemērs*

Šī datu kopa satur 100000 krāsainus attēlus, sadalītus 200 klasēs (pa 500 attēliem priekš katrai klasei) un samazinātus līdz 64x64 pikseļiem. Katra klase satur 500 apmācības, 50 validācijas un 50 testa attēlus.

### 2.6.3 Attēli ar cilvēkiem medicīniskās maskās

*Attēls 18: "People wearing masks" datu kopnes attēlu piemērs*

Tā ir salīdzinoši augstas izšķirtspējas datu kopa (79698 failu, 165 GB apjomā), kuras attēli tiks samazināti un apgriezti līdz kopējam 512x512 izmēram.

### 2.6.4 Pašveidota transporta attēlu kopa

Autora pašizveidota augstas izšķirtspējas (9504x6336, 61Mpix) attēlu kopa.

Bija pieņemts lēmums izveidot savu kopu, jo tad ir pieeja RAW sākumformātam, no kura var ģenerēt jebkura formāta attēlus. Kas nav iespējams, izmantojot stipri apgrieztus, samazinātus un kompresētus 8 bitu JPEG failus.

## 2.7 Testēšanas protokols

### Datorsistēmas konfigurācija

CPU: AMD Ryzen Threadripper 2950X

GPU: Nvidia RTX A4000 16GB (PCIe 3.0 x16 slotā)

RAM: 4 kanālu 64GB (8x8) DDR4 3200MHz @2933MHz

SSD: Samsung 970 EVO Plus 4TB (NVME, 2x2TB RAID 0)

### Programatūras konfigurācija

Operētājsistēma: Ubuntu 22.10 / Linux 5.4

Python: 3.11.1

PyTorch: 1.13.1+cu116

Katrai metodei nepieciešams izmērīt:

- Kopējais apmācības laiks
- Vidējais epohas laiks
- Vidējais epohas datu ielādes laiks
- Maksimālais RAM patēriņš
- Maksimālais VRAM patēriņš
- GPU noslodze
- CPU noslodze

# 3 Rezultāti

Datu ielādes metodes PRET datu kopām tabulās.

# 4 Tālākie pētījumi

# 5 Secinājumi

# 6 Bibliogrāfija


[1] I. Vasilev, D. Slater, G. Spacagna, P. Roelants, un V. Zocca, *Python deep learning: exploring deep learning techniques and neural network architectures with PyTorch, Keras, and TensorFlow*, Second edition. Birmingham Mumbai: Packt Publishing Limited, 2019.

[2] N. Buduma un N. Locascio, *Fundamentals of deep learning: designing next-generation machine intelligence algorithms*, First edition. Sebastopol, CA: O'Reilly Media, 2017.

[3] "Binary crossentropy loss function | Peltarion Platform". https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/binary-crossentropy (skatīts 2022. gada 19. aprīlī).

[4] C. Albon, *Machine learning with Python cookbook: practical solutions from preprocessing to deep learning*, First edition. Sebastopol, CA: O'Reilly Media, 2018.

[5] C. C. Aggarwal, *Neural networks and deep learning: a textbook*. Cham, Switzerland: Springer, 2018. doi: 10.1007/978-3-319-94463-0.

[6] "CuPy", *CuPy*. https://cupy.dev/ (skatīts 2022. gada 19. aprīlī).

[7] *DLPack: Open In Memory Tensor Structure*. Distributed (Deep) Machine Learning Community, 2022. Skatīts: 2022. gada 19. aprīlī. [Tiešsaiste]. Pieejams: https://github.com/dmlc/dlpack

[8] "NVIDIA Magnum IO GPUDirect Storage Overview Guide". http://docs.nvidia.com/gpudirect-storage/overview-guide/index.html (skatīts 2022. gada 20. aprīlī).

[9] "DirectStorage API Now Available on PC", *DirectX Developer Blog*, 2022. gada 14. martā. https://devblogs.microsoft.com/directx/directstorage-api-available-on-pc/ (skatīts 2022. gada 20. aprīlī).

[10] "DALI", *NVIDIA Developer*, 2019. gada 5. martā. https://developer.nvidia.com/dali (skatīts 2022. gada 20. aprīlī).