МАТЕМАТИЧКА ГИМНАЗИЈА

МАТУРСКИ РАД

ИЗ РАЧУНАРСТВА И ИНФОРМАТИКЕ

Конволутивне неуронске мреже

Аутор:

Алекса Јоцковић IVe

Ментор:

проф. Милош Арсић

Београд, мај 2025.

Садржај

1. УВОД 1

1.1 Увод у машинско учење 1

1.2 Главне категорије машинског учења 2

2. НЕУРОНСКЕ МРЕЖЕ 5

2.1 Грађа вештачког неурона у неуронским мрежама 6

2.2 Улазни слој у неуронским мрежама 9

2.3 Скривени слојеви у неуронским мрежама 11

2.4 Излазни слој у неуронским мрежама 12

3. КАКО НЕУРОНСКА МРЕЖА УЧИ 13

3.1 Функције грешке 13

3.3 Појам градијентног спуста 14

3.2 Пропагација унапред и пропагација уназад 16

3.4 Оптимизациони алгоритми 18

3.4.1 Стратегије градијентног спуста 18

3.4.2 Напредни оптимизациони алгоритми 19

3.5 Регуларизација у неуронским мрежама 22

3.6 *Transfer learning* и *finetuning* 24

3.7. Евалуација модела и метричке функције 24

4. КОНВОЛУТИВНЕ НЕУРОНСКЕ МРЕЖЕ (CNN) 26

4.1 Архитектура CNN-а 26

4.2 Конволутивни слојеви 27

4.3 *Pooling* слојеви 30

4.4 Примена CNN-а 32

4.5 Савремене CNN архитектуре и еволуција 33

5. ИМПЛЕМЕНТАЦИЈА CNN-а 35

5.1 Најзначајније датотеке 36

5.2 Структура и резултати имплементације 37

5.3 Комплетна анализа Python кода: CNN и класификација слика 38

6. ЗАКЉУЧАК 41

7. ЛИТЕРАТУРА 42

1. УВОД

1.1 Увод у машинско учење

Машинско учење (енгл. *Machine Learning*, ML) представља грану вештачке интелигенције која се бави развојем алгоритама и модела који омогућавају рачунарима да самостално уче из података и доносе одлуке без експлицитног програмирања. Машинско учење данас има кључну улогу у многим областима, као што су медицина, финансије, транспорт, роботика и комуникације. У савременом друштву, количина доступних података експоненцијално расте, а традиционалне методе анализе постају неадекватне. Управо ту машинско учење преузима примат, јер пружа алате за аутоматизацију анализе великих скупова података и препознавање комплексних образаца. Машинско учење може се дефинисати као процес у којем рачунарски системи користе алгоритме за идентификацију образаца у подацима и користе их за доношење предикција или одлука. Уместо да буду експлицитно програмирани да изврше одређени задатак, системи уче из примера.

Први конкретни алгоритми за учење развијени су током 1950-их и 1960-их. На пример, *Perceptron,* који је представио Frank Rosenblatt 1958. године, био је први модел неуронске мреже способан да „учи“ из података (Rosenblatt, 1958). Иако су тадашњи модели били једноставни и ограничених могућности, поставили су темеље за даље истраживање.

У деценијама које су уследиле, машинско учење се развијало паралелно са развојем рачунара, статистике и теорије информација. Током 1980-их и 1990-их, појављују се снажнији алгоритми као што су стабла одлучивања, методе најближих суседа, *Bayes*-ови класификатори и методе подршке векторима (SVM), који су омогућили боље перформансе и ширу примену (Mitchell, 1997). Кључни допринос у овој фази дао је Том Mitchell, који је 1997. године дефинисао машинско учење као „науку која проучава алгоритме који омогућавају рачунарима да уче из искуства“ (Mitchell, 1997).

Велики пробој догодио се почетком 21. века захваљујући порасту количине доступних података (тзв. *big data*) и значајном напретку у рачунској моћи. То је омогућило развој дубоког учења (енгл. *deep learning*), које користи вишеслојне неуронске мреже за решавање сложених проблема као што су препознавање говора, слика и језика (LeCun, Bengio, & Hinton, 2015). Данас, машинско учење је неизоставан алат у многим научним и индустријским дисциплинама.

У Табели 1 приказани су кључни догађаји у развоју машинског учења.

Табела 1. Кључни догађаји у развоју машинског учења

Година	**Догађај / Допринос**	**Име / Институција**
1950.	Предлог Тјуринговог теста – питање „Могу ли машине мислити?“	Alan Turing
1957.	Развој перцептрона – првог модела неуронске мреже	Frank Rosenbaltt
1967.	Почетак развоја алгоритма *k*-најближих суседа (KNN)	Evelyn Fix & Joseph Hodges
1986.	Популаризација алгоритма *backpropagation* за тренирање неуронских мрежа	Rumelhart, Hinton & Williams
1995.	Увођење метода подршке векторима (CBM) у праксу	Vapnik & Cortes
1997.	Формална дефиниција машинског учења као поља	Tom Mitchell
2006.	Почетак модерне ере дубоког учења (*deep learning*)	Geoffrey Hinton
2012.	Револуција у препознавању слика – *AlexNet* побеђује на *ImageNet* такмичењу	Krizhevsky, Sutskever & Hinton
2016.	DeepMInd-ов *AlphaGo* побеђује светског шампиона у игри *Go*	DeepMind (Google)
2020+	Употреба великих језичких модела (нпр. GPT) у обради природног језика	OpenAI

1.2 Главне категорије машинског учења

Машинско учење обухвата више категорија које се разликују према типу проблема и типу доступних података:

1. Надгледано учење (*Supervised Learning*)

У надгледаном учењу, модел се тренира на унапред означеним подацима. Сваком улазном примеру додељен је одговарајући излаз, а циљ модела је да научи функцију која што боље предвиђа излаз за нове, непознате податке. Ова врста учења најчешће се користи када имамо јасно дефинисан скуп података са познатим исходима. Примери:

- **Класификација** – када модел предвиђа којој класи припада пример (нпр. препознавање да ли је слика пас или мачка).
- **Регресија** – када модел предвиђа нумеричку вредност (нпр. предикција цене куће на основу површине, локације и других фактора).

2. Ненадгледано учење (*Unsupervised Learning*)

Ненадгледано учење користи податке који нису унапред означени. Модел покушава да открије скривене обрасце, структуре или односе у подацима без експлицитних излазних вредности. Ова врста учења се често користи у анализи великих скупова података како би се идентификовали природни кластери или димензије. Примери:

- **Груписање (кластеровање)** – као што је сегментација корисника на основу понашања (нпр. *online* купци са сличним навикама).
- **Смањење димензионалности** – као што је визуализација високо-димензионалних података.

3. Учење поткрепљивањем (*Reinforcement Learning*)

У учењу поткрепљивањем, модел (агент) учи кроз интеракцију са окружењем, при чему на основу својих акција добија повратне информације у облику награда или казни. Циљ агента је да максимизира укупну награду кроз учење стратегије доношења одлука. Ова метода се посебно користи у областима где су одлуке секвенцијалне и утичу на будуће стање система. Примери:

- Обука робота да хода или избегава препреке.
- Аутоматизовано играње игара (нпр. *AlphaGo,* који је савладао људске шампионе у игри *Go*).
- Оптимизација токова у саобраћају или финансијским портфељима.

У пракси се најчешће користе алгоритми као што су линеарна регресија, стабла одлучивања, неуронске мреже и методе подршке векторима (Hastie, Tibshirani, & Friedman, 2009). Посебно су популарни дубоки модели засновани на вештачким неуронским мрежама, који су постигли изванредне резултате у обради слике и природног језика (LeCun, Bengio, & Hinton, 2015). Развој ових метода уско је повезан са доступношћу великих количина података и рачунске снаге (Domingos, 2012). Важан део сваког машинског модела јесте процес тренирања и евалуације, у којем се користи скуп познатих података како би се модел усмерио ка што тачнијим предикцијама (Bishop, 2006).

Иако машинско учење доноси бројне предности, као што су аутоматизација процеса, побољшана ефикасност и нова сазнања из података, важно је обратити пажњу на етичка питања и ризике погрешне примене (Crawford & Calo, 2016). Један од кључних проблема јесте **пристрасност у подацима** (*data bias*), која може довести до

дискриминаторних одлука ако тренирани модели рефлектују неједнакости присутне у историјским подацима. На пример, алгоритми за запошљавање или кредитно одобравање могу нехотице фаворизовати или искључивати одређене друштвене групе уколико тренирају на небалансираним скуповима података.

Поред тога, постоји забринутост због **недостатка транспарентности,** тзв. „црна кутија“ проблема, где чак ни креатори модела не могу у потпуности објаснити зашто је модел донео одређену одлуку. Ово поставља питање одговорности, нарочито у осетљивим областима као што су здравство, правосуђе или безбедност. Још један значајан изазов је **нарушавање приватности**, јер системи машинског учења често обрађују велике количине личних података, понекад без експлицитне сагласности корисника. Уз то, могућност злоупотребе алгоритама за надзор или манипулацију информацијама, као што се дешава са тзв. *deepfake* технологијама, додатно истиче потребу за регулацијом и етичким смерницама. Као што истичу Crawford и Calo (2016), истраживачи и инжењери често занемарују шири друштвени контекст технологије коју развијају, што може довести до озбиљних последица по једнакост, слободу и поверење јавности. Због тога је важно да се развој машинског учења одвија у складу са принципима **праведности, одговорности и транспарентности** (тзв. FAIR принципи – *Fairness, Accountability, and Transparency*).

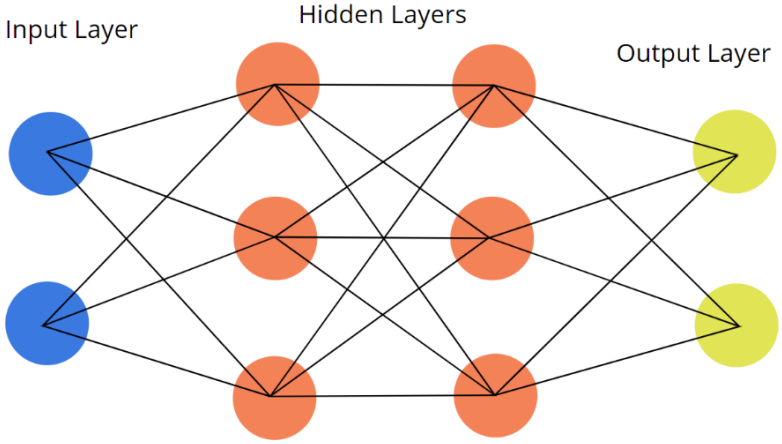
Један од најзначајнијих приступа у оквиру машинског учења, посебно у последњој деценији, јесу **вештачке неуронске мреже.** Представљају систем повезаних јединица – неурона – које заједнички обрађују податке и „уче“ обрасце из сложених улаза. Захваљујући способности да моделирају нелинеарне односе и открију скривене структуре у великим скуповима података, неуронске мреже су постале основа за многе савремене технологије, укључујући препознавање слика, обраду природног језика и аутономна возила. У наставку ће бити објашњени основни елементи неуронских мрежа, начин њиховог учења и разлике у односу на класичне алгоритме машинског учења.

2. НЕУРОНСКЕ МРЕЖЕ

У савременом добу вештачке интелигенције, **неуронске мреже** представљају једно од најмоћнијих оруђа за решавање сложених задатака из области као што су препознавање слика, говор, обрада језика и предикција података. Инспирисане начином на који функционише људски мозак, неуронске мреже се састоје од међусобно повезаних **вештачких неурона** (Слика 1) који уче из података кроз процес који подсећа на учење код људи. Њихова способност да откривају скривене обрасце у великим скуповима података чини их основом многих напредних технологија данашњице.

Слика 1. Повезани неурони

Основна структура неуронске мреже састоји се од три типа слојева, Слика 2, **улазног слоја** који прима податке (*input layer, blue*), **скривених слојева** (*hidden layers, red*) који врше обраду информација и **излазног слоја** (*output layer, green*) који даје крајњи резултат. Сваки слој се састоји од неурона који су повезани међусобно преко тежина и активационих функција. Кроз процесе **пропагације унапред** и **пропагације уназад**, мрежа прилагођава своје параметре како би побољшала тачност учења.



Слика 2. Основна структура неуронске мреже

Улазни слој (*Input layer*):

Улазни слој прима сирове податке који улазе у мрежу. Ови подаци могу бити слике, текст, нумеричке вредности, или било који други облик информација које мрежа треба да процесуира. Сваки неурон у улазном слоју представља један атрибут или карактеристику скупа података. На пример, за препознавање слика, сваки пиксел слике може бити улазни податак.

Скривени слојеви (*Hidden layers*):

Скривени слојеви обављају већину рачунских операција у мрежи. Они су одговорни за учење сложених образаца у подацима. Сваки неурон у скривеним слојевима прима улазе од неурона у претходном слоју, примењује одговарајуће тежине, користи активацијску функцију (као што су *ReLU*, *sigmoid* или *tanh*) и шаље резултат следећем слоју. Скривени слојеви омогућавају мрежи да моделира нелинеарне односе у подацима и ефикасно препозна сложене образце.

Излазни слој (*Output layer*):

Излазни слој даје крајњи резултат или предикцију. На основу претходних слојева и учења, излазни слој генерише одговоре у зависности од врсте проблема. На пример, код проблема класификације, излазни слој може имати један неурон за сваку класу и користи функцију попут *softmax* за генерисање вероватноћа за сваку класу. Код проблема регресије, излаз може бити једна бројчана вредност.

Постоји више врста неуронских мрежа, у зависности од њихове архитектуре и намене. Најосновнији облик је **вишеслојна перцептронска мрежа (MLP)**, која се састоји од више слојева потпуно повезаних неурона. **Конволутивне неуронске мреже (CNN)** користе се првенствено за обраду слика и визуелних информација, јер су способне да препознају локалне обрасце као што су ивице, облици и текстуре. С друге стране, **рекурентне неуронске мреже (RNN)** специјализоване су за обраду секвенцијалних података, као што су временске серије или природни језик, јер омогућавају памћење претходних информација. Захваљујући разноликости архитектура и флексибилности у раду са различитим врстама података, неуронске мреже су постале темељ модерног машинског учења и један од најперспективнијих праваца развоја вештачке интелигенције.

2.1 Грађа вештачког неурона у неуронским мрежама

Основна градивна јединица неуронске мрезе је **неуронски чвор**, који се често назива једноставно **неурон**. Иако је инспирисан биолошким неуронима у људском мозгу,

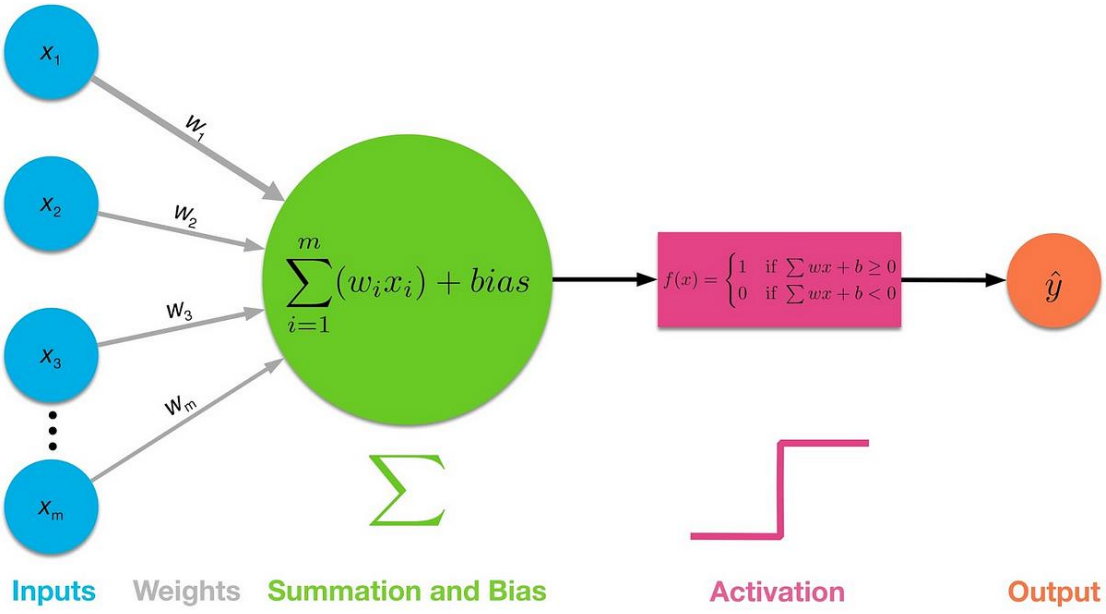
вештачки неурон је поједностављен математички модел који има јасно дефинисану структуру и функцију. Разумевање грађе и улоге неурона кључно је за схватање начина на који неуронске мреже функционишу, уче и доносе одлуке. Вештачки неурон се састоји од три основне компоненте, Слика 3:

1. Улази (*Inputs*)

Неурон прима више улазних сигнала који долазе из спољног света (нпр. пиксела слике, сензорских података, нумеричких вредности) или из других неурона претходног слоја мреже. Сваки улаз означава се као x_i, где је i индекс улазне компоненте.

2. Тежине (*Weights*)

Сваки улаз је упарен са реалним бројем који се назива **тежина** и означава се као w_i. Тежине представљају значај који неурон придаје одређеном улазу. Веће тежине значе већи утицај улаза на крајњу одлуку неурона.



Слика 3. Основна структура неурона

3. Сума и активациона функција (комбиновање и активација)

Неурон сабира све улазе помножене одговарајућим тежинама и додаје тзв. биас. **Биас** (помак) је додатни параметар у вештачком неурону који омогућава мрежи да боље моделира податке. Он функционише као константа која се додаје на суму улаза помножених тежинама, пре него што се резултат проследи активационој функцији, Слика 3. Математички, излаз неурона се рачуна као:

$$z = \sum_{i=1}^{n} w_i \cdot x_i + b \qquad (1)$$

где је z **линеарни збир** (улаз у активациону функцију), w_i су **тежине** које одређују значај сваког улаза, x_i су **улазне вредности** (спољашњи свет или претходни слој неурона), b је **биас** (помак) и представља константну вредност.

Затим се на ову суму примењује **активациона функција** ϕ, која одлучује да ли ће се неурон "активирати" (односно дати излазну вредност) или не. Математички израз за излаз неурона може се записати и као:

$$z = \sum_{i=1}^{n} w_i \cdot x_i + b \Rightarrow a = \phi(z) \tag{2}$$

где је a излазна вредност неурона.

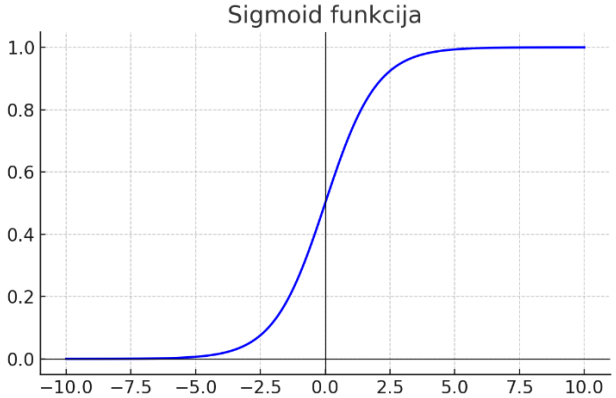
Активациона функција даје **нелинеарност** неуронској мрежи, што је од пресудног значаја за њену способност да учи сложене обрасце и нелинеарне односе у подацима. Без активационе функције, неуронска мрежа би се сводила на низ линеарних трансформација, што би значајно ограничило њену изражајну моћ, без обзира на број слојева.

Активационе функције такође омогућавају **скалирање излаза неурона** на одређени опсег, као што је између **0 и 1** (код *sigmoid* функције), или **–1 и 1** (код *tanh* функције), чиме се постиже стабилније и ефикасније учење. Ова нормализација помаже при конвергенцији модела током тренирања, нарочито када се користе методе попут градијентног спуста. Неке од најчешће коришћених активационих функција су:

***Sigmoid* функција:**

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3}$$

Користи се код бинарних класификација, компресујући вредности у опсег (0,1) што се може интерпретирати као вероватноћа, Слика 4. Међутим, код већих мрежа може довести до проблема избледњивања градијената (*vanishing gradient problem*).

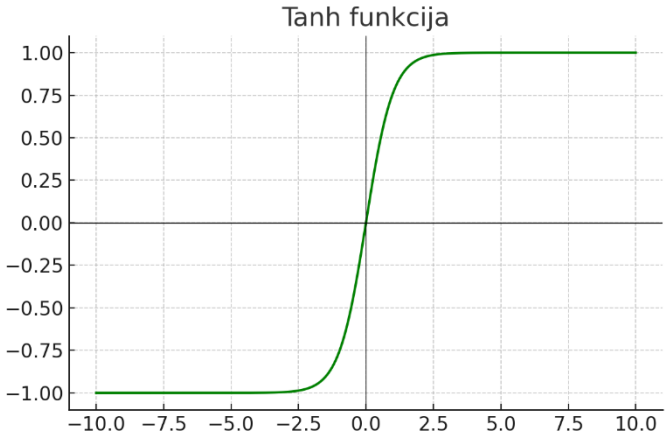


Слика 4. *Sigmoid* функција

Tanh (хиперболички тангенс)

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{4}$$

Скалира вредности у опсег (–1,1) и често даје боље резултате од сигмоида у скривеним слојевима, јер је центрирана око нуле, Слика 5.

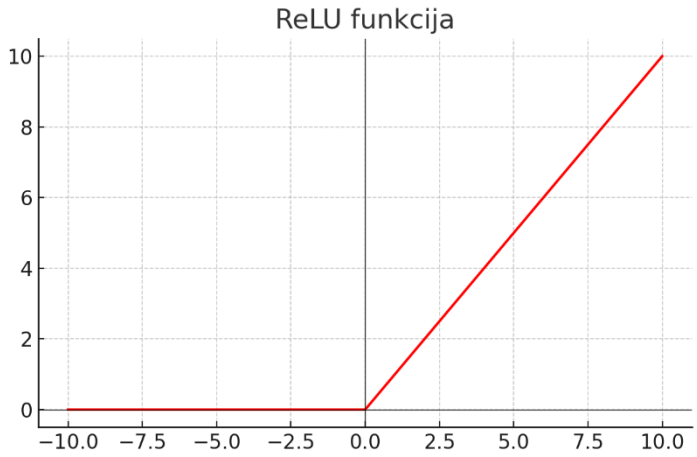


Слика 5. Tanh функција

ReLU (*Rectified Linear Unit*)

$$\mathrm{ReLU}(z) = max(0, z) \tag{5}$$

Најчешће коришћена функција у модерним мрежама. Једноставна је и ефикасна, али може довести до тзв. **мртвих неурона** када су вредности константно негативне, Слика 6.



Слика 6. ReLU функција

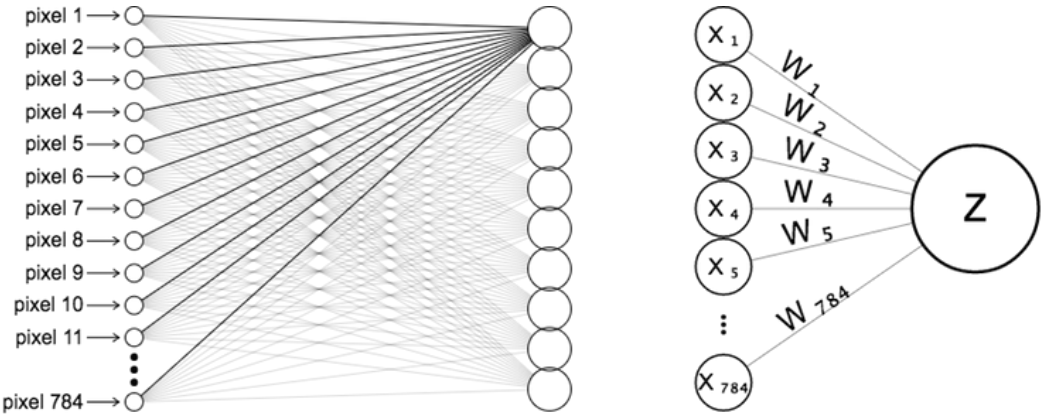
2.2 Улазни слој у неуронским мрежама

Улазни слој представља **почетну тачку** у структури сваке вештачке неуронске мреже. Његова основна улога је да **прими сирове податке** из спољног света и **проследи их**

даље у дубље слојеве мреже на обраду. Без улазног слоја, неуронска мрежа не би могла да комуницира са окружењем и не би имала информације на основу којих учи. Улазни слој се састоји од неурона који не врше никакве математичке трансформације над подацима, тј. не врше обраду нити прилагођавају податке. Они служе као пасивни преносиоци вредности, односно улога је једноставно да их „прикључе“ на неуронску мрежу. Сваки неурон у улазном слоју одговара једној компоненти улазног податка:

- За слике: сваки пиксел слике мапира се на један неурон
- За векторе: свака нумеричка вредност мапира се на посебан неурон
- За текст: улазни текст се често кодира у бројеве (*embedding*), па сваки број иде у један неурон.

На пример, за слику димензија 28×28 пиксела (784 пиксела укупно), улазни слој ће имати 784 неурона, при чему сваки неурон носи вредност интензитета одговарајућег пиксела, Слика 7.



Слика 7. Улазни слој

Главна функција улазног слоја је:

- Примање података из спољног света (нпр. слика, звук, текст, бројеви)
- Презентација података мрежи у облику који може да се обради
- Организација података тако да буду доступни скривеним слојевима.

Структура и димензије улазног слоја зависе од типа и формата улазних података:

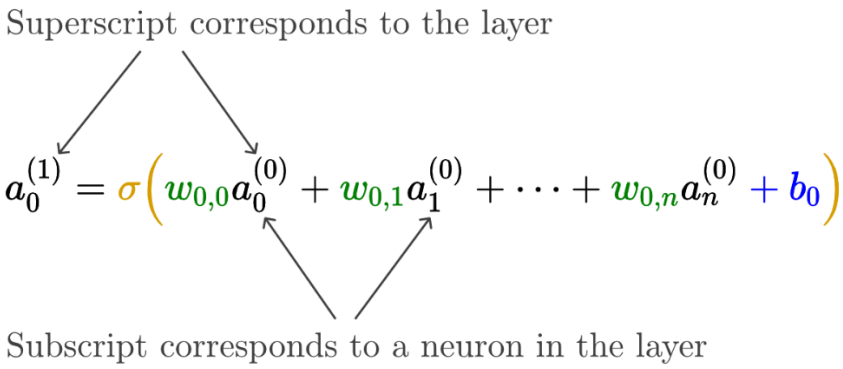
- За слике: користи се тродимензионални улаз (висина, ширина, број канала), нпр. *32×32×3* за RGB слике.
- За секвенцијалне податке: користи се временска димензија, нпр. низ бројева за временске серије.
- За табуларне податке: користи се једноставан вектор са бројем неурона једнаким броју атрибута.

На тај начин, улазни слој омогућава мрежи да се прилагоди различитим врстама проблема и апликација.

2.3 Скривени слојеви у неуронским мрежама

У архитектури вештачких неуронских мрежа, **скривени слојеви** имају кључну улогу у способности мреже да учи, препознаје обрасце и доноси тачне предикције. Скривени слојеви представљају све слојеве неурона који се налазе **између улазног** и **излазног слоја** и називају се „скривеним“ јер њихови излази нису директно доступни кориснику. Видљиве су само информације на улазу и излазу мреже.

Сваки скривени слој састоји се од једног или више неурона, при чему је сваки неурон повезан са свим неуронима из претходног слоја (у случају потпуно повезаних мрежа), или са одређеним локалним регионима (у конволутивним мрежама). Математички, активација једног неурона у скривеном слоју може се записати као (Слика 8):



Слика 8. Излазна вредност неурона

Број скривених слојева и број неурона у сваком слоју значајно утичу на капацитет неуронске мреже:

- **Мали број скривених слојева** (тзв. плитке мреже) може бити довољан за једноставне проблеме са линеарним или слабим нелинеарним односима.
- **Велики број скривених слојева** формира **дубоке неуронске мреже** (*Deep Neural Networks – DNN*), које су способне да уче изузетно комплексне обрасце, као што су препознавање лица, аутоматски превод језика или детекција објеката на слици.

Међутим, повећање броја слојева и неурона може довести до проблема као што су **пренаученост** (*overfitting*) и **експлозија/гашење градијената**, па се морају користити додатне технике као што су регуларизација, *dropout* или *batch* нормализација.

2.4 Излазни слој у неуронским мрежама

Излазни слој представља завршну компоненту сваке вештачке неуронске мреже. Његова основна функција је да на основу претходно научених карактеристика кроз скривене слојеве произведе коначну **предикцију** или **одлуку**. Кроз излазни слој, мрежа саопштава своје резултате кориснику или другом систему који користи ове информације. Излазни слој се састоји од једног или више неурона, а број неурона директно зависи од типа задатка који мрежа решава:

- **За бинарну класификацију**: излазни слој има 1 неурон (нпр. "да" или "не").
- **За вишекласну класификацију**: број неурона одговара броју класа (нпр. за препознавање цифара 0–9 користи се 10 неурона).
- **За регресионе задатке**: излазни слој може имати један неурон који даје реалну вредност (нпр. предвиђање цене куће).

Сваки неурон у излазном слоју обрађује информације из последњег скривеног слоја и производи резултат који интерпретирамо као мрежину излазну предикцију. За разлику од скривених слојева, активациона функција у излазном слоју се пажљиво бира у зависности од природе проблема:

- ***Sigmoid* функција** (*prethodno opisana)*.
- ***Softmax* функција**: користи се у вишекласној класификацији, нормализује вектор излаза тако да све вредности буду у опсегу (0,1) и збир свих излаза буде 1.
- **Линеарна функција (без активације)**: користи се код регресије, где излаз може бити било који реалан број.

Правилно одабрана активациона функција омогућава тачно и смислено интерпретирање резултата мреже. Активационе функције у излазном слоју се углавном разликују од активационих функција које се користе у скривеним слојевима. Примери излазних слојева у пракси:

- Детекција присуства објекта (да ли постоји пешак на слици): користи се 1 неурон са *sigmoid* активацијом.
- Препознавање руком писаних цифара (0–9): користи се 10 неурона са *softmax* функцијом.
- Предвиђање температуре: користи се 1 неурон без активације или са линеарном активацијом.

3. КАКО НЕУРОНСКА МРЕЖА УЧИ

Учење представља централни процес у функционисању вештачких неуронских мрежа. Кроз учење, мрежа постепено прилагођава своје унутрашње параметре (тежине и биас вредности) са циљем да минимизује грешку у предикцијама. Основни циљ учења је да неуронска мрежа на основу доступних примера из тренинг скупа развије способност општег закључивања, односно тачне предикције на невиђеним подацима.

Када неуронска мрежа прими улазне податке, она их пропагира кроз своје слојеве и производи излаз, односно предикцију. Та предикција се затим упоређује са стварном циљаном вредношћу, а разлика између њих квантификује се помоћу функције грешке (*loss function*). Функција грешке представља један од кључних елемената у процесу тренирања неуронских мрежа.

У процесу смањења грешке, значајну улогу има **градијент**, који представља вектор парцијалних извода функције грешке у односу на параметре мреже (тежине и биас вредности). Градијент показује правац и брзину у којем функција грешке најбрже расте, што значи да указује како треба да се промене параметри мреже да би се смањила грешка. На основу градијента, неуронска мрежа користи **алгоритам градијентног спуста** како би ажурирала параметре. Дакле, градијент омогућава мрежи да "разуме" како се промена сваког параметра одражава на грешку модела. На тај начин, кроз итеративне кораке, неуронска мрежа постепено побољшава своје перформансе на основу анализираних података, све док не постигне жељени ниво тачности у предикцијама.

3.1 Функције грешке

Крос-ентропијска функција грешке (*Cross-entropy loss*)

Крос-ентропијска функција грешке се широко користи у проблемима класификације, нарочито када се на излазу неуронске мреже користи *softmax* функција која резултира расподелом вероватноће по класама.

Нека је:

$\boldsymbol{y} \in \{0,1\}^C$ вектор стварних вредности (*one-hot* кодиран)

$\overline{\boldsymbol{y}} \in [0,1]^C$ вектор предикција модела, добијен *softmax* активацијом

тако да важи:

$$\sum_{i=1}^{C} \bar{y}_i = 1 \tag{6}$$

тада је крос-ентропијска функција грешке дефинисана као:

$$LCE(y, \bar{y}_i) = \sum_{i=1}^{C} -y_i \cdot \log(\bar{y}_i) \tag{7}$$

У бинарној класификацији (када је број класа $C = 2$), она се поједностављује у облик:

$$L_{BCE}(y, \bar{y}) = -[y \cdot \log(\bar{y}) + (1 - y) \cdot \log(1 - \bar{y})] \tag{8}$$

Ова функција снажно кажњава погрешне, а сигурне класификације.

Средњеквадратна грешка (*Mean Squared Error – MSE*)

Најчешће коришћена у регресионим проблемима, средњеквадратна грешка рачуна квадрат разлике између предикција и стварних вредности:

$$L_{MSE}(y, \bar{y}) = \left(\frac{1}{n}\right) \sum_{i=1}^{n} (y_i - \bar{y}_i)^2 \tag{9}$$

Глатка је и диференцијабилна, што омогућава стабилну оптимизацију помоћу градијентних метода. Међутим, осетљива је на екстремне вредности (*outlier*-е).

Средње апсолутна грешка (*Mean Absolute Error – MAE*)

Алтернатива MSE функцији је средње апсолутна грешка, дефинисана као:

$$L_{MAE}(y, \bar{y}) = \left(\frac{1}{n}\right) \sum_{i=1}^{n} (|y_i - \bar{y}_i|) \tag{10}$$

MAE је мање осетљива на *outlier*-е у односу на MSE, али није глатка у тачки $y_i = \bar{y}_i$ што може отежати конвергенцију.

Хуберова функција грешке (*Huber loss*)

Хуберова функција комбинује предности MSE и MAE, понаша се као квадратна грешка за мале разлике, а као апсолутна за велике:

$$L\delta(y, \bar{y}) = \begin{cases} \frac{1}{2}(y - \bar{y})^2, & ako\ |y - \bar{y}| \leq \delta \\ \delta\left(|y - \bar{y}| - \frac{1}{2}\delta\right), & inače \end{cases} \tag{11}$$

Параметар δ одређује праг између два режима понашања.

3.3 Појам градијентног спуста

Градијентни спуст (*Gradient Descent*) је један од најважнијих алгоритама за **оптимизацију** у математици и примењеним наукама. Његова основна сврха је

проналазак минимума функције више променљивих, посебно када експлицитно решење минимизације није доступно аналитичким путем. Градијентни спуст је основа многих нумеричких метода у областима као што су статистика, обрада сигнала, економија и инжењеринг. У контексту неуронских мрежа, градијентни спуст омогућава проналажење оптималних вредности тежина и биас вредности које минимизују функцију грешке. Градијент функције грешке у односу на параметре мреже, односно градијент, показује правац у којем би требало да се помере тежине и биас вредности како би се смањила грешка. Формално, задатак оптимизације се своди на минимизацију функције:

$$\min_{x \in \mathbb{R}^n} f(x) \tag{12}$$

где је:

- $f: \mathbb{R}^n \rightarrow \mathbb{R}$ – функција чији минимум тражимо,
- x – вектор променљивих $x = (x_1, x_2, \dots, x_n)$.

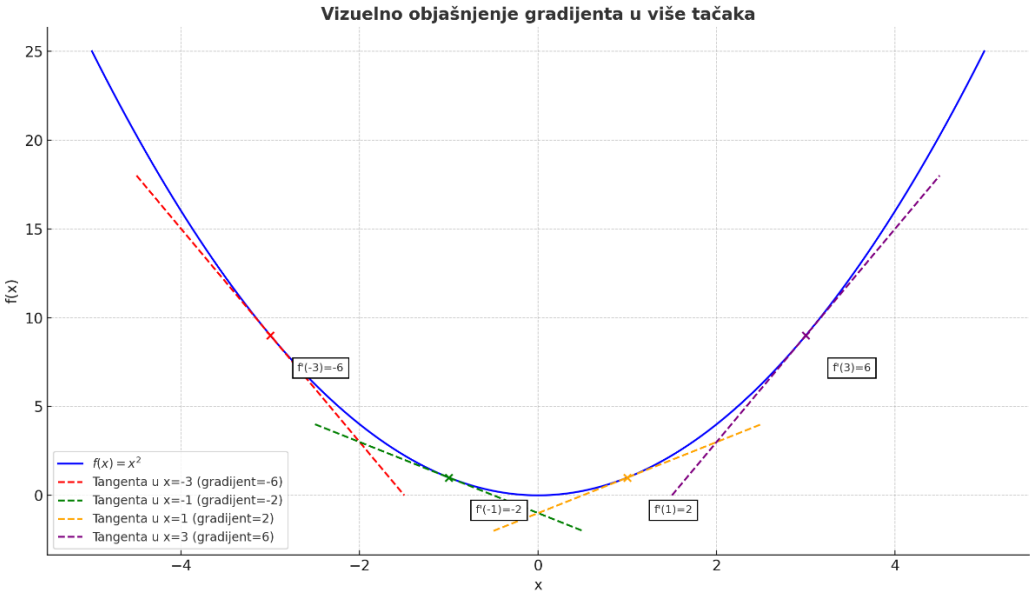
Циљ је пронаћи вредност x^* за коју је $f(x^*)$ најмања могућа.

$$f(x^*) \leq f(x) \text{ за свако } x \in \mathbb{R}^n \tag{13}$$

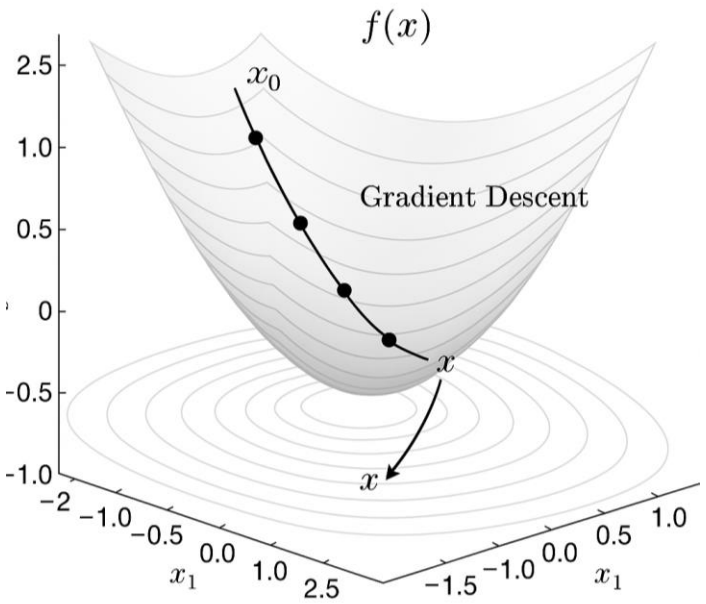
Градијент функције $f(x)$ означен као $\nabla f(x)$ представља вектор који показује смер најбржег раста функције. Да бисмо смањили вредност функције, природно је кретати се супротно смеру градијента. Кретање низ градијент може се замислити као кретање ка дну криве. На Слици 9 је приказан поступак градијентног спуста у 2D простору са две променљиве. Линије на слици представљају контуре функције грешке. Свака тачка представља један корак у оптимизацији, док стрелице показују како се алгоритам креће кроз простор параметара у правцу смањења грешке. Циљ алгоритма је да дође до центра, где је грешка најмања, а то је **глобални минимум**. Градијентни спуст користи информације о нагибу (тј. изводима) да би у сваком кораку прилагодио параметре и приближио се том минимуму.

На Слици 10 приказана је визуелна интерпретација градијентног спуста на конвексној функцији у облику параболоида. Црне тачке представљају итерације алгоритма, почевши од почетне тачке x_0, па све до минимума функције у тачки x. Стрелица и крива линија означавају правац кретања алгоритма, који се на сваком кораку помера у супротном смеру од градијента функције, тражећи најнижу тачку. Контур-линије на дну представљају изолине функције, тј. линије једнаких вредности, док облик визуелизује како функција расте удаљавањем од минимума. Ова слика

омогућава интуитивно разумевање како градијентни спуст "корак по корак" минимизује грешку померајући се низбрдо ка дну функције.



Слика 9. Пример градијентног спуста



Слика 10. Визуелна интерпретација градијентног спуста

3.2 Пропагација унапред и пропагација уназад

Вештачке неуронске мреже функционишу кроз два основна механизма: **пропагацију унапред** (*forward propagation*) и **пропагацију уназад** (*back propagation*). Пропагација унапред представља процес у којем мрежа, на основу улазних података, пролази кроз

низ слојева и рачуна излазну вредност (предикцију). У овој фази, информације „теку“ од улаза ка излазу мреже, пролазећи кроз тежине и активационе функције сваког неурона. Након што мрежа произведе излаз, он се пореди са стварном циљаном вредношћу помоћу функције грешке. Циљ је да се ова грешка минимизира, што омогућава учење. Ту наступа пропагација уназад, алгоритам који омогућава мрежи да „научи“ из грешке и представља кључни алгоритам учења у вештачким неуронским мрежама. Пропагација уназад користи ланчану правило диференцирања (*chain rule*) како би израчунала како сваки параметар (тежина и биас) утиче на укупну грешку. На основу тих информација, мрежа прилагођава своје параметре тако да следећи пут направи бољу предикцију. Ова два процеса заједно чине основни циклус учења у неуронским мрежама и омогућавају им да моделују сложене нелинеарне односе у подацима.

У првом кораку, **пропагација унапред**, мрежа прима улазне податке, рачуна активације сваког неурона кроз све слојеве и производи излазну предикцију. Након што се предикција упореди са циљаном вредношћу помоћу функције грешке, мрежа мора да прилагоди своје параметре како би смањила грешку.

У другом кораку **пропагацијом уназад** се рачунају градијенти функције грешке узимајући у обзир све тежине и биас вредности у мрежи, чиме омогућава прилагођавање параметара у правцу смањења грешке. У практичним условима, рачунање градијената функционише тако што се вредности излаза и парцијалних извода чувају за сваки слој током пропагације унапред, а затим се користе рекурзивно у пропагацији уназад. Циљ пропагације уназад је да одговори на питање: "Како промена сваке појединачне тежине и биас вредности утиче на укупну грешку мреже?" Процес пропагације уназад може се поделити у три главне фазе:

1. **Израчунавање грешке на излазу:** прво се израчунава разлика између стварне циљне вредности и предикције на излазу мреже. За сваки неурон у излазном слоју, грешка се изражава као парцијални извод функције грешке у односу на излазну активацију.
2. **Ширење грешке уназад кроз мрежу:** грешке из излазног слоја се пропагирају назад кроз слојеве мреже. Сваки неурон у скривеним слојевима прима грешке од својих излазних веза и користи их за израчунавање сопствене грешке.
3. **Рачунање градијената и ажурирање тежина:** на основу израчунате грешке, за сваки параметар (тежину и биас) израчунава се градијент – тј. смер и интензитет

промене потребне за смањење грешке. Параметри се потом ажурирају коришћењем правила градијентног спуста.

Свака функција мреже је композиција више функција, дефинисана као:

$$f(x) = f_L\left(f_{L-1}(\dots f_1(x))\right) \tag{14}$$

где је сваки f_L једна трансформација – обично линеарна функција тежина и улаза, са нелинеарном активационом функцијом. Да бисмо израчунали како промена параметра у раном слоју утиче на крајњи излаз мреже, користимо ланчано правило диференцирања:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial f_L} \cdot \frac{\partial f_L}{\partial f_{L-1}} \cdot \dots \cdot \frac{\partial f_1}{\partial x} \tag{15}$$

У практичним условима, рачунање градијената функционише тако што се вредности излаза и парцијалних извода чувају за сваки слој током пропагације унапред, а затим се користе рекурзивно у пропагацији уназад.

Овај циклус – пропагација унапред, израчунавање грешке, пропагација уназад и ажурирање тежина – понавља се током тренинга све док мрежа не постигне задовољавајућу тачност.

3.4 Оптимизациони алгоритми

Оптимизација у неуронским мрежама представља кључни део процеса тренирања и представља проналажење најефикасније примене градијентног спуста. Неуронске мреже садрже велики број параметара, који контролишу како се улазни подаци трансформишу и користе за предвиђање. Оптимизација омогућава да се ови параметри ефикасно прилагоде тако да минимизују функцију грешке између излаза модела (предикција) и стварних вредности.

Један од најважнијих аспеката оптимизације у учењу је метода која се користи за ажурирање параметара мреже током тренирања. Постоји неколико стратегија оптимизације, односно градијентног спуста, а свака има своје предности и мане у зависности од специфичних захтева модела и скупа података.

3.4.1 Стратегије градијентног спуста

1. ***Batch* градијентни спуст (*Batch Gradijent Descend BGD*)**

   Овај приступ користи целокупан скуп података за ажурирање параметара мреже. Процес се састоји у томе да се рачунају градијенти функције грешке за све узорке у скупу података и онда се изврши једно ажурирање параметара. Иако је врло стабилан, јер се користи свеобухватна информација о подацима, BGD је врло спор,

нарочито када су скупови података веома велики, јер захтева значајну рачунарску снагу и меморију за рад са целокупним скупом података у сваком кораку.

2. **Стохастички градијентни спуст (*Stochastic Gradient Descent SGD*)**

   Стохастички градијентни спуст ажурира параметре мреже након обраде сваког појединачног узорка (*data point*). Ово омогућава брже ажурирање параметара и ефикаснији рад са великим скуповима података. Међутим, овај приступ може довести до великих осцилација у процесу тренирања, јер се параметри мењају након сваког узорка, што може отежати постизање конвергенције. Иако је бржи, SGD није увек стабилан и може се "осциловати" око оптималних решења.

3. ***Mini-batch* градијенти спуст (*Mini-batch Gradient Descend MBGD*)**

   Mini-batch градијентни спуст комбинује предности оба претходна приступа. Уместо да користи целокупан скуп података (BGD) или само један узорак (SGD), користи мале групе узорака (*mini-batches*) за ажурирање параметара. Обично се користи број узорака између 32 и 256, што омогућава брзо рачунање градијената уз задржавање стабилности у процесу тренирања. *Mini-batche* приступ балансира брзину и стабилност, јер смањује осцилације у процесу тренирања које настају у SGD, док истовремено омогућава бржу обрада података него BGD.

Свака од ових метода има своје предности у различитим ситуацијама, а избор одговарајуће стратегије зависи од специфичних карактеристика проблема и скупа података. У пракси, MBGD се често користи, јер нуди добар компромис између брзине и стабилности током процеса тренирања.

3.4.2 Напредни оптимизациони алгоритми

Напредни оптимизациони алгоритми играју кључну улогу у процесу тренирања дубоких неуронских мрежа, јер омогућавају брже конвергирање, стабилност и ефикасност приликом ажурирања параметара мреже. Коришћењем различитих техника за адаптацију стопе учења, ови алгоритми могу да превазиђу ограничења класичног градијентног спуста, као што су спора конвергенција и осцилације. Стопа/брзина учења (*learning rate*, означава се ознаком η) представља величину корака који се предузима у правцу супротном од градијента функције грешке, приликом ажурирања параметара мреже. У сваком кораку градијентног спуста, нова вредност параметра рачуна се по формули:

$$w_{(t+1)} = w_t - \eta \cdot \nabla L(w_t) \tag{16}$$

где је:

- η стопа (брзина) учења,
- L функција грешке (енгл. *loss function*),
- $\nabla L(w_t)$ градијент функције грешке
- w_{t+1} тежине из следећег корака итерације
- $t, t+1$ представљају број итерације

Правилно подешена брзина учења омогућава да се мрежа ефикасно приближава минимуму функције грешке, без непотребног осциловања или застоја. Изабрана висока вредност η може изазвати дивергенцију, односно уместо да се мрежа приближава минимуму, она "прескаче" оптималне вредности, а грешка може чак расти уместо да опада. У графичком смислу, мрежа се понаша нестабилно и удаљава се од минимума. Ниска вредност η води до веома спорог учења, односно мрежа се креће ка минимуму, али врло малим корацима, што значајно продужава време тренирања и често води у локалне минимуме из којих се тешко излази.

Следећи напредни оптимизациони алгоритами су широко примењени у тренирању дубоких мрежа, посебно у контексту конволуционих неуронских мрежа (CNN):

1. *Momentum*

Momentum је техника која убрзава стохастички градијентни спуст (SGD) додавањем "моментума" на основу претходних градијената. Ова техника помаже да се превазиђу локални минимуми и стабилизује процес тренирања. Пример израчунавања моментума за тежину:

$$v_t = \gamma \cdot v_{t-1} + \eta \cdot \nabla L(w_{t-1}) \quad (17)$$

$$w_t = w_{t-1} - v_t \quad (18)$$

где је:

- v_t брзина или акумулирани градијент на тренутном кораку
- γ фактор моментума, обично између 0 и 1 (типично 0.9)
- η стопа (брзина) учења,
- L функција грешке (енгл. *loss function*),
- $\nabla L(w_{t-1})$ градијент функције грешке у односу на тежине на претходном кораку.
- w_{t-1} тежине из претходног корака итерације
- $t, t-1$ представљају број итерације

Momentum омогућава брже конвергирање, јер користи претходне градијенте за "убрзање" тренирања, смањујући број корака потребних за постизање минимума.

2. *Nesterov momentum*

Nesterov momentum унапређује класични моментум гледајући испред тренутне позиције, што доводи до прецизнијег предвиђања смерова кретања параметара. Формуле су следеће:

$$v_t = \gamma \cdot v_{t-1} + \eta \cdot \nabla L(w_{t-1} - \gamma \cdot v_{t-1}) \quad (19)$$

$$w_t = w_{t-1} - v_t \quad (20)$$

Овај метод омогућава брже и прецизније кораке, јер "гледа унапред" и чини корекцију према оптимизованој тачки, чиме се смањује број корективних корака.

3. RMSProp (*Root Mean Square Propagation*)

RMSProp је алгоритам који адаптира величину корака за сваки параметар узимајући у обзир квадрате недавних градијената. Овај алгоритам користи покретни просек квадрата градијената, чиме помаже да се избегну велики кораци у правцу "осцилација" у подручју где је градијент мали, док омогућава бржи напредак у правцу где је градијент велики. Формуле су:

$$E[g^2]_t = \rho \cdot E[g^2]_{t-1} + (1 - \rho) \cdot g^2 \quad (21)$$

$$w_t = w_{t-1} - \frac{(\eta \cdot g)}{\sqrt{E[g^2]_t + \varepsilon}} \quad (22)$$

где је:

- $E[g^2]_t$ експоненцијални покретни просек квадрата градијената
- ρ фактор смањења, обично постављен на 0.9
- ε мали број који спречава дељење са нулом (типично 10^{-8}).

RMSProp је посебно ефикасан у тренирању модела са великим бројем параметара, јер помаже у адаптацији стопе учења на основу статистике градијената.

4. Adam (*Adaptive Moment Estimation*)

Adam је један од најпопуларнијих оптимизационих алгоритама и комбинује предности RMSProp и *momentum*-а. Он одржава експоненцијално покретне просеке првог (m) и другог (v) момента градијента, омогућавајући брже и стабилније ажурирање параметара. Adam користи следеће формуле:

1. $m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (23)
   (експ. просек првог момента – средња вредност градијента)

2. $v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (24)
   (експ. просек другог момента – квадрат градијента)
3. $\hat{m}_t = m_t / (1 - \beta_1^t)$ (25)
   (корекција пристрасности за m)
4. $\hat{v}_t = v_t / (1 - \beta_2^t)$ (26)
   (корекција пристрасности за v)
5. $w_t = w_{t-1} - \eta \cdot \hat{m}_t / \sqrt{(\hat{v}_t + \varepsilon)}$ (27)
   (ажурирање тежина)

где су:

- β_1 *i* β_2 – фактори експоненцијалног смањења (*decay rate*) за моменат првог и другог реда, β_1 је обично 0.9, β_2 је обично 0.999
- $\hat{m}_t$ *i* $\hat{v}_t$ – корекције пристрасности (*bias correction*): користе се у раним фазама тренирања јер су m_t и v_t иницијално близу нуле.

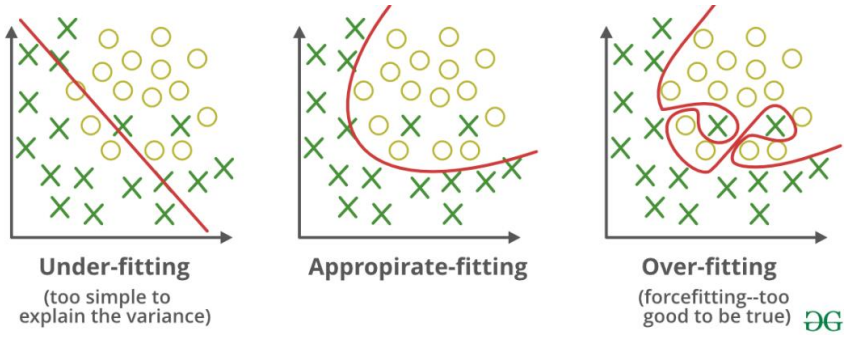
Вредност првог момента некад није мала због поништавања услед промена правца, већ просто зато што је норма градијента мала, иако је правац стабилан. У таквој ситуацији дељење малом оценом другог момента води убрзавању кретања што је увек пожељно ако нема промена правца. У случају великих градијената, чак и кад се осцилује, резултујућа вредност може бити велика. Дељење великом оценом другог момента води смањењу корака и бржем заустављању осцилација. Adam је најчешће коришћен у раду са конволуционим неуронским мрежама (CNN), јер омогућава стабилну, аутоматски прилагодљиву стопу учења и брзо конвергирање. Због својих својстава, Adam је врло ефикасан за рад са великим скуповима података и сложеним архитектурама.

Напредни оптимизациони алгоритми омогућавају значајна побољшања у стабилности, брзини и ефикасности тренирања дубоких неуронских мрежа. Прилагодљиве стопе учења, као и експоненцијални покретни просеци, омогућавају ефикасније оптимизовање сложених модела на великим скуповима података, чинећи ове технике неопходним у савременом учењу машина.

3.5 Регуларизација у неуронским мрежама

Регуларизација представља скуп техника које се користе да би се спречило **прекомерно прилагођавање** (*overfitting*) модела тренирајућим подацима. До *overfitting*-а долази када модел не научи општа правила, већ почне да „памти“ конкретне примере из тренинг скупа, укључујући и шум, тј. случајне и нерелевантне

информације, Слика 11. Такав модел има ниску грешку на тренирајућем скупу, али слабије перформансе на невиђеним подацима, што значи да лоше генерализује.



Слика 11. Прекомерно прилагођавање (*overfitting*)

CNN модели, због своје велике дубине и огромног броја параметара, нарочито су осетљиви на *overfitting*, посебно када се тренирају на малим, неуравнотеженим или шумовитим скуповима података. У таквим условима, модел може превише зависити од детаља из тренинг скупа, што доводи до смањења његове способности да правилно класификује нове примере.

Најчешће технике регуларизације у неуронским мрежама укључују:

- ***Dropout*** – метода у којој се насумично искључује одређени проценат неурона у току обуке. Тиме се спречава да поједини неурони постану превише доминантни и модел се учи да не зависи од специфичних активација. *Dropout* се обично примењује у потпуно повезаним слојевима.
- **L2 регуларизација (*Weight decay*)** – додаје се „кажњавањем“ великих вредности тежина у функцији грешке. Циљ је да се тежине одрже мале, чиме се смањује комплексност модела и повећава његова способност генерализације.
- ***Batch Normalization*** – техника која се користи за стабилизацију и убрзавање процеса учења у дубоким неуронским мрежама. Њена основна идеја је да се нормализују улази сваког слоја, односно да се њихова средња вредност постави на нулу, а стандардна девијација на један. Тиме се постиже да сви слојеви у мрежи добијају податке у сличном распону, што значајно ублажава проблем нестајућих или експлодирајућих (превише малих и превише великих) градијената.

Ове технике често се комбинују у модерним неуронским архитектурама како би се постигла максимална робусност и стабилност током обуке.

3.6 *Transfer learning* и *finetuning*

Transfer learning (учење преносом) представља технику у дубоком учењу којом се користи већ тренирани модел на новом задатку. Ова метода је изузетно корисна када немамо велики скуп података за обучавање или желимо да убрзамо процес учења. Уместо да се модел тренира из почетка, користи се знање (параметри) научено на великом и општем скупу података као што је *ImageNet* и преноси се на специфичнији задатак. Најчешћи приступ трансфер учењу у конволутивним неуронским мрежама јесте коришћење унапред истренираног модела као што су: *VGG*, *ResNet*, *Inception* или *MobileNet*, и његово прилагођавање новом задатку класификације, детекције или сегментације. Постоји неколико стратегија како се то ради:

- **Замрзавање слојева** (*feature extraction*): замрзну се сви слојеви осим последњег (или неколико последњих) скривеног слоја, а на врх се додаје нови класификатор који се тренира на циљаном скупу података. Ово је корисно када су нови подаци слични онима на којима је модел претходно трениран.
- ***Finetuning***: поред замене класификатора, дозвољава се делимично тренирање и дубљих слојева мреже. То омогућава да се модел боље прилагоди специфичностима новог задатка. *Finetuning* захтева пажљиво подешавање стопе учења и може донети значајно побољшање перформанси.

Предности *transfer learning*-а су вишеструке:

- значајно смањење времена тренирања,
- боља генерализација,
- мање потребе за великим скуповима података,
- ефикасније искоришћење рачунарских ресурса.

Ова техника се широко користи у медицинској дијагностици, препознавању објеката, класификацији слика и другим доменима где је прикупљање и анотација података скупа или технички захтевна. Савремене библиотеке као што су *TensorFlow* и *PyTorch* пружају унапред истриниране моделе који се лако могу користити у трансфер учењу.

3.7. Евалуација модела и метричке функције

Евалуација перформанси модела заснива се на различитим метричким функцијама које квантификују тачност класификације, Слика 11. Најчешће коришћене мере укључују прецизност (*precision*), одзив (*recall*) и F1 скор (*F1 score*).

- **Прецизност** представља проценат тачних позитивних предикција у односу на све позитивне предикције које је модел изнео:

$$Precision = TP/(TP + FP) \quad (28)$$

где је *TP* (*true positive*) број истинских позитивних, а *FP* (*false positive*) број лажно позитивних случајева. Висока прецизност значи да када модел класификује нешто као позитивно, велика је вероватноћа да је то заиста тачно.

- **Одзив** представља проценат тачно детектованих позитивних примера у односу на укупан број стварних позитивних примера:

$$Recall = TP/(TP + FN) \quad (29)$$

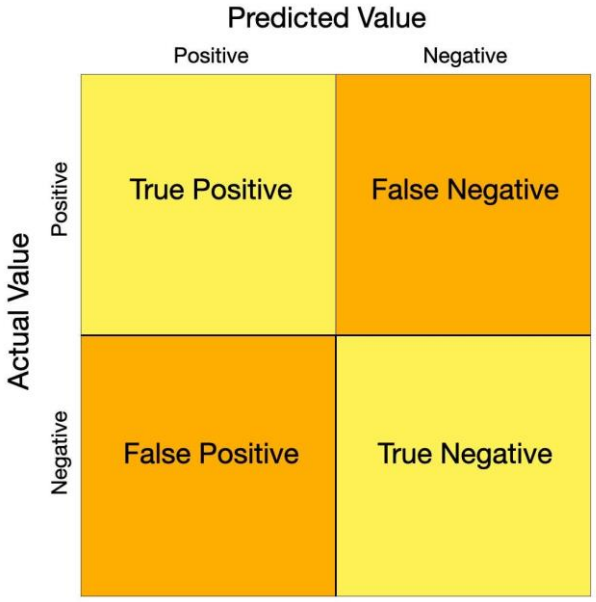
где је *FN* (*false negative*) број лажно негативних примера. Одзив мери способност модела да "ухвати" све позитивне случајеве.

- **F1 скор** представља хармонијску средину између прецизности и одзива, и користи се када је потребан баланс између ова два критеријума.

$$F1 = 2 \cdot \frac{(Precision \cdot Recall)}{(Precision + Recall)} \quad (30)$$

F1 скор је посебно користан када имамо неуравнотежен скуп података и када нам је подједнако важно да модел не пропушта позитивне случајеве (висок *recall*), али ни да не даје превише лажних аларма (висок *precision*).

Ове метричке функције омогућавају прецизнију евалуацију модела (Слика 12) од саме тачности (*accuracy*), нарочито када се примењују на домене са значајно неуравнотеженим класама као што су медицинска дијагностика, безбедност и детекција грешака.



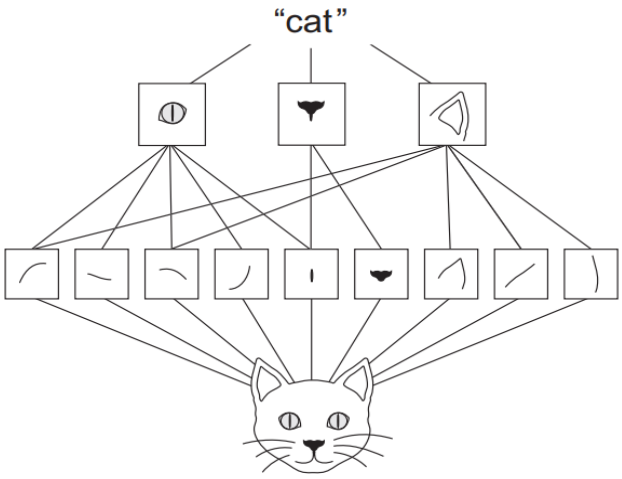
Слика 12. Евалуација модела

4. КОНВОЛУТИВНЕ НЕУРОНСКЕ МРЕЖЕ (CNN)

Конволутивне неуронске мреже (CNN) су дубоки модели који се истичу способношћу да аутоматски уче обрасце из сирових података. Њихова архитектура је инспирисана начином на који визуелни кортекс у мозгу процесуира информације – помоћу локалних рецептивних поља и хијерархијске обраде. CNN се примењују у многим савременим технологијама и истраживањима, од аутономне вожње до биомедицинске анализе и постале су централни стуб у развоју вештачке интелигенције.

CNN су специјално осмишљене за обраду података са локалном и просторном структуром, као што су слике, аудио и видео. За разлику од класичних потпуно повезаних неуронских мрежа, CNN користи локално повезане слојеве и конволутивне филтере који омогућавају аутоматско препознавање локалних образаца и значајно смањење броја параметара, чиме се повећава ефикасност учења у савременом дубоком учењу, а нарочито у области рачунарског вида.

Једна од кључних особина CNN-а јесте способност хијерархијског учења репрезентација. Нижи слојеви у CNN-у препознају једноставне шаре (нпр. хоризонталне и вертикалне ивице), док виши слојеви комбинују те шаре у сложеније обрасце, као што су контуре објеката, делови лица или чак специфични предмети као што су аутомобили или птице, Слика 13.



Слика 13. Визуелно представљање хијерархијског учења CNN-а

4.1 Архитектура CNN-а

Архитектура конволутивне неуронске мреже састоји се од низа специфичних слојева који трансформишу улазне податке у репрезентације корисне за решавање задатка, као што је класификација слике. Кључни елементи CNN архитектуре укључују, Слика 14:

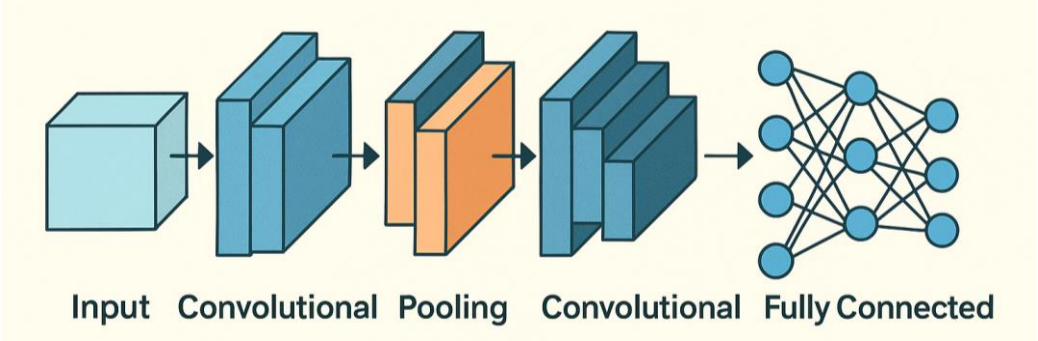
1. Улазни слој

2. Конволутивни слојеви (*Convolutional layers*)

4. Слојеви сажимања (*Pooling layers*)

5. Неуронска мрежа (*Fully Connected layers*)

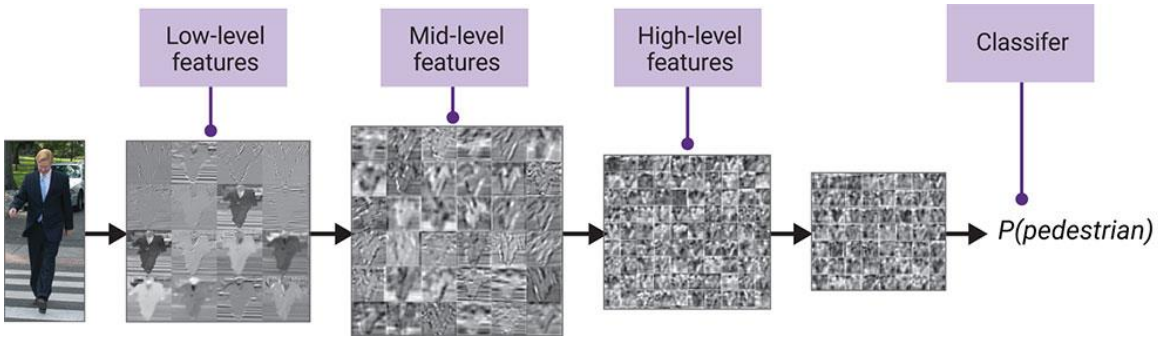
6. Излазни слој



Слика 14. Кључни елементи CNN архитектуре

4.2 Конволутивни слојеви

Конволутивни слојеви су срж CNN архитектуре. У њима се примењују тзв. филтери или кернели, који проклизавају (*slide*) преко улазне слике и примењују операцију конволуције како би се издвојиле локалне карактеристике, као што су ивице, углови и текстуре. Сваки филтер детектује одређени тип обрасца. Што се дубље иде у мрежу, то слојеви детектују комплексније и семантички богатије карактеристике. Резултат конволуције је *feature map* или мапа карактеристика. Њена сврха је да открије и сачува важне локалне обрасце из података, Слика 15.



Слика 15. *Feature map* пример

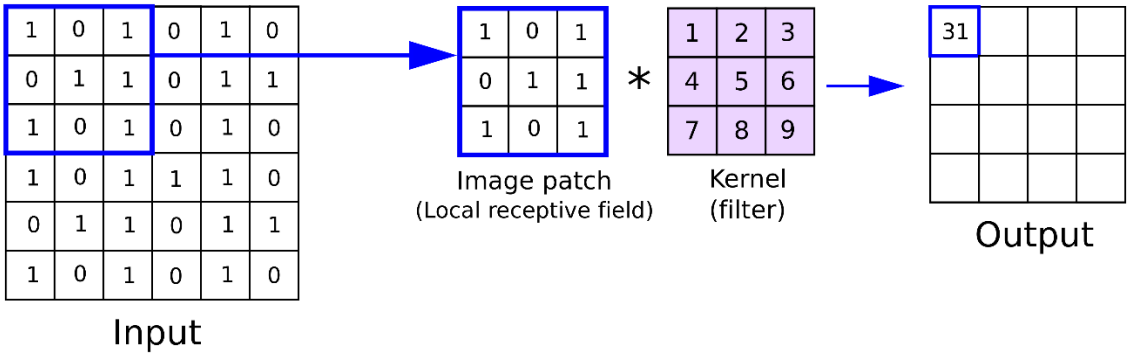
Конволутивни слој функционише применом математичке операције познате као **конволуција**. У контексту неуронских мрежа, то је у ствари „пролажење“ мањих матрица (тзв. филтера или језгара) преко улазне слике или активационе мапе претходног слоја, Слика 16. На свакој позицији се израчунава скаларни производ између филтера и одговарајућег дела слике, чиме се добија нова вредност на излазу. Математички, конволуција између филтера K и улазне слике I се рачуна као:

$$S(i,j) = \Sigma_m \Sigma_n I(i+m, j+n) \times K(m,n) \quad (31)$$

где је:

- $S(i,j)$ резултат конволуције на позицији *(i, j)*
- $I(i + m, j + n)$ вредност пиксела из улазне слике
- $K(m,n)$ вредност филтера (језгра, кернела)
- m, n индекси који се крећу кроз димензије филтера.

У пракси, сваки филтер детектује специфичан образац – један може да открива хоризонталне ивице, други вертикалне, трећи текстуре итд.



Слика 16. *Feature map* процедура

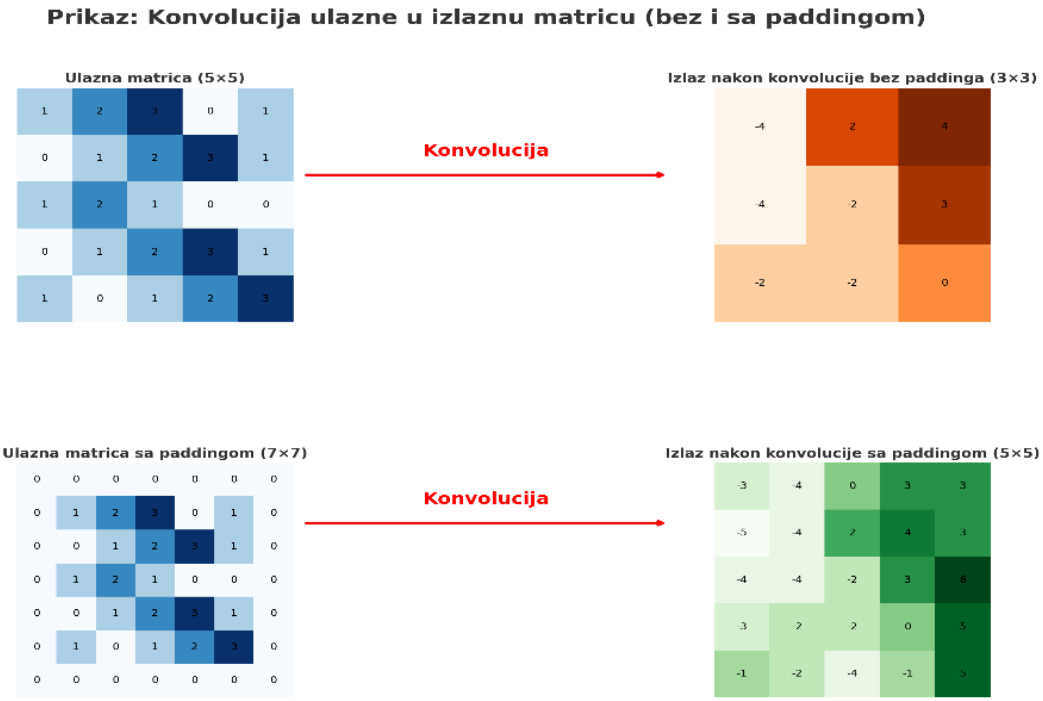
Конволутивни слој се дефинише помоћу више важних параметара:

1. **Број филтера (језгра):** сваки филтер у слоју се понаша као детектор одређеног обрасца и генерише посебну *feature map*-у. Ако слој садржи *n* филтера, произвешће *n* мапа карактеристика, односно *n* канала на излазу.
2. **Величина филтера (*kernel size*):** Најчешће коришћене величине су 3×3, 5×5 и 7×7. Мањи филтери, попут 3×3, омогућавају градњу дубљих мрежа уз мањи број параметара, чиме се постиже боља ефикасност и прецизније учење.
3. ***Stride* (корак):** *Stride* означава број пиксела за који се филтер помера при сваком кораку преко слике.
   - *Stride* = 1: филтер се помера пиксел по пиксел, што резултира већом излазном димензијом.
   - Већи *stride* (нпр. 2) смањује димензије излаза и може служити као облик *downsampling*-а (смањење количине информација).
4. ***Padding* (попуњавање):** *Padding* се користи како би се контролисала величина излазне мапе. То је техника којом се око улазне слике додају додатни редови и колоне (најчешће нуле) пре него што се на њу примени конволуциони филтер. Главна сврха *padding*-а је да очува димензије излаза након конволуције, омогући

да филтери обраде и ивице слике, спречи пребрзо смањивање димензионалности података кроз мрежу. Постоји више врста *padding*-а:

- ***Valid padding* (без *padding*-а):** у овом режиму не додаје се ништа око ивица слике. Конволуција се примењује само на оне делове улаза на које филтер у потпуности може да се „наслони“, тј. валидне позиције. Димензије излазне мапе карактеристика (*feature map*) се **смањују** у односу на улаз. Погодно када се жели постепено смањење димензија слике и екстракција хијерархијских особина.
- ***Same padding* (истих димензија):** овај приступ користи додавање (обично нуле) тако да се димензије излазне мапе задрже исте као код улаза (када је *stride*=1). Нарочито је корисно када је важно задржати исту резолуцију дуж већег броја слојева у мрежи. Омогућава дубље мреже без превеликог губитка информација.
- ***Zero-padding* (попуњавање нулама),** најчешћи технички приступ *padding*-у. Око улазне слике се додају нуле да би филтер могао да „види“ и ивице слике. Тиме се спречава да информације са ивица буду изгубљене у каснијим слојевима. Омогућава већи број излазних пиксела и равномернију обраду целе слике

На пример, улаз димензије 5×5 са филтером 3×3 без *padding*-а резултира излазом 3×3. Ако се користи *padding* ширине 1, излаз остаје 5×5, Слика 17.



Слика 17. Конволуција улазне у излазну матрицу

4.3 *Pooling* слојеви

Pooling (обједињавање, сажимање) слојеви се примењују непосредно након конволутивних слојева и њихова основна сврха је сажимање локалних информација. У типичном CNN-у, низ који чине конволутивни и *pooling* слој представља основну јединицу структуре мреже, а затим се та секвенца понавља више пута у дубљим нивоима. Главне функције *pooling* слојева су:

- **Редукција димензионалности** (*downsampling*): смањују се просторне димензије (висина и ширина) података, чиме се смањује број параметара и рачунска сложеност.
- **Екстракција доминантних карактеристика**: бирају се најрепрезентативније информације из регија слике.
- **Повећање транслационе инваријантности**: модел постаје отпорнији на мале транслације (померање) улазног објекта.
- **Превенција *overfitting*-а**: редуковањем информација смањује се шанса да модел „научи напамет“ специфичне детаље из тренинг сета.

Најчешће коришћене *pooling* операције су:

1. Максимални *pooling* (*Max Pooling*):

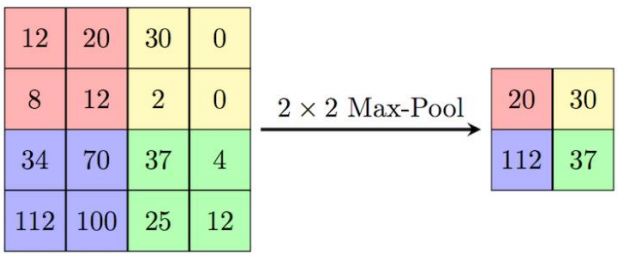
Најзаступљенији тип *pooling* операције. Максимални *pooling* издваја највећу вредност из сваке подрегије улазне мапе активације. Математички се *pooling* може писати као:

Нека је $R \subset \mathbb{R}^{nxn}$ – регион (прозор) из улазне мапе карактеристика над којим се примењује поолинг операција. Тада је излаз *Max Pooling* слоја дефинисан као:

MaxPooling(R) = максимум од свих вредности x које припадају региону R, односно:

MaxPooling(R) = *max(x)*, за све $x \in R$ (32)

Ова операција задржава само највећу вредност из сваког локалног прозора, чиме се мрежа фокусира на најизраженије карактеристике у подацима, Слика 18.



Слика 18. Визуализација *max pooling-а*

2. Просечни *pooling* (*Average Pooling*)

Уместо максимума, израчунава се просечна вредност у подрегији. Ова метода задржава више информација, али губи на изражавању доминантних карактеристика. Математички се *pooling* може писати као:

Нека је $R \subset \mathbb{R}^{nxn}$ – регион (прозор) из улазне мапе карактеристика над којим се примењује поолинг операција. Тада је излаз дефинисан као:

AveragePooling (R) = $(1 \,/\, |R|)$ × сума свих x који припадају R, (33)

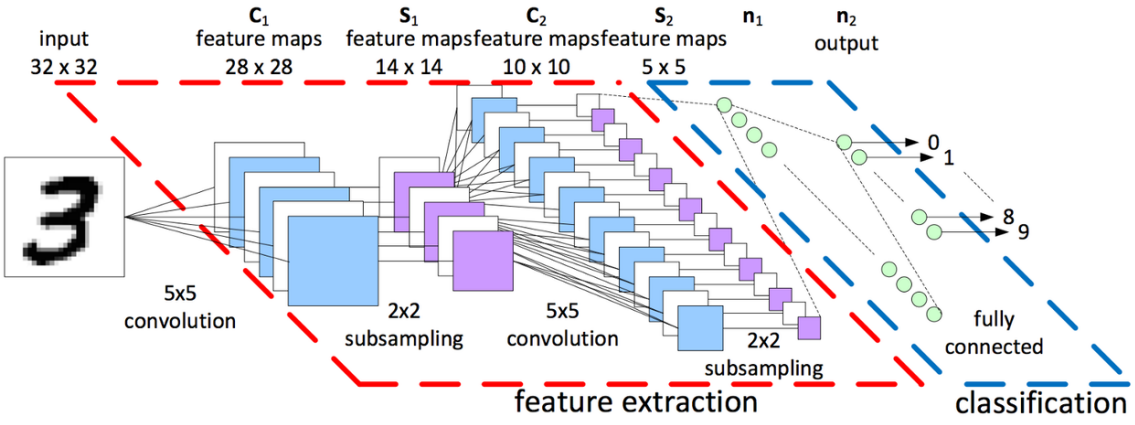
AveragePooling (R) = $(1/\text{број елемената у } R) \times \sum x$, за свако $x \in R$ (34)

3. Глобални *pooling* (*Global Max / Average Pooling*)

Глобални *pooling* је операција која се примењује над целом активационом мапом сваког канала, обично непосредно пре излазног слоја мреже. За разлику од стандардног *pooling* -а који се примењује над мањим регијама (нпр. 2×2 или 3×3), глобални *pooling* обједињује све вредности у целокупној мапи и своди их на једну јединy вредност по каналу.

На пример, ако је активациона мапа димензија 7×7, глобални *pooling* ће произвести само једну вредност по каналу, без обзира на величину мапе. Глобални *Max Pooling* узима максималну вредност из целог канала. Глобални *Average Pooling* рачуна просечну вредност свих активација у каналу.

Све претходне наведене компоненте CNN-а могу се сумирати у Слици 19 која представља класификацију ручно написаног броја 3.

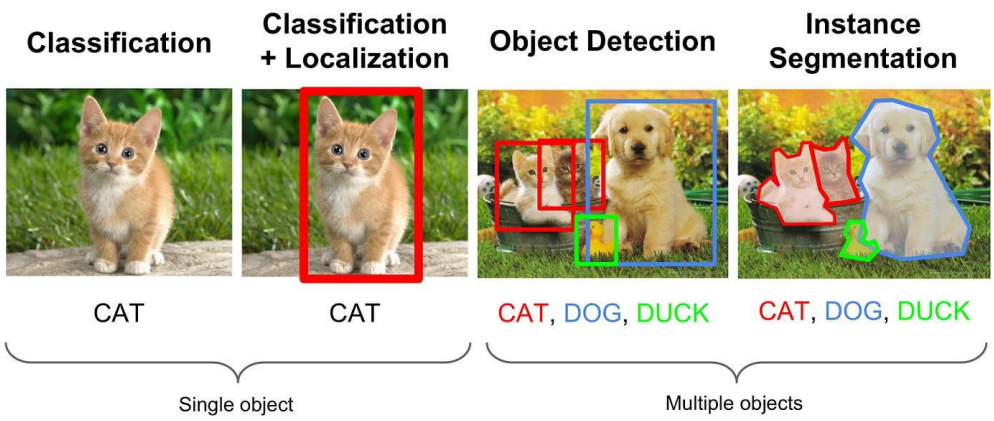


Слика 19. Класификација ручно написаног броја 3

4.4 Примена CNN-а

Конволутивне неуронске мреже (CNN) се широко примењују у задацима који захтевају разумевање визуелних информација, као што су слике и видео. Кључне примене укључују, Слика 20:

1. **Класификација (*Classification*):** CNN модели се успешно користе за бинарне и вишекласне класификационе задатке, где је циљ да се одреди да ли одређени објекат постоји на слици или којој класи слика припада. На пример, у бинарној класификацији, мрежа одлучује да ли на слици постоји пас или не. Код вишекласне класификације, модел идентификује којој од више унапред дефинисаних класа слика припада (нпр. пас, мачка, аутомобил итд.).
2. **Локализација (*Localization*):** поред утврђивања присуства објекта, **локализација** се односи на одређивање **тачне позиције објекта у слици**. То се обично постиже регресијом координата (*bounding box*) које дефинишу правоугаоник око објекта. CNN се у овом случају допуњује додатним излазима који предвиђају локацију објекта заједно са класом.
3. **Детекција објеката (*Object Detection*):** Детекција комбинује класификацију и локализацију тако да модел открива више објеката у слици, класификује их и предвиђа њихове позиције. Архитектуре као што су R-CNN, *fast* R-CNN, *faster* R-CNN и YOLO (*You Only Look Once*) интегришу ове функционалности у ефикасан *end-to-end* систем.
4. **Сегментација (*Segmentation*):** за дубље разумевање слике, CNN модели се користе за семантичку сегментацију, где се сваком пикселу слике додељује припадност одређеној класи (нпр. сви пиксели који припадају аутомобилу, дрвету, небу итд.). Напреднији модели, као што су U-Net и Mask R-CNN, омогућавају прецизну сегментацију објеката, чак и у сложеним сценама.



Слика 20. Разумевање визуелних информација

Најпознатија и најранија употреба CNN-а била је у класификацији руком писаних цифара на MNIST скупу података, где су CNN модели показали супериорне перформансе у поређењу са класичним алгоритмима. У ширем смислу, CNN су се показале као темељ за све модерне приступе рачунарском виду.

У области **медицинске дијагностике**, CNN модели се користе за аутоматску детекцију абнормалности на рендгенским, CT и MRI снимцима. На пример, тренирањем CNN мреже на великом броју означених снимака, модел може научити да препознаје тумор, фрактуре или патолошке промене са тачношћу блиском људској.

У **аутономним возилима**, CNN мреже су одговорне за интерпретацију визуелних података из камера постављених на возилу – укључујући детекцију трака, знакова, препрека и других возила. То омогућава возилу да „види“ и реагује у реалном времену, што је кључно за безбедно управљање.

У **индустрији безбедности**, CNN се користи за препознавање лица, идентификацију особа на основу надзорних снимака, па чак и за анализу израза лица како би се детектовало потенцијално сумњиво понашање.

CNN су се показале и као основа за **генеративне моделе**, укључујући стилизацију слика, супер-резолуцију, па чак и генерисање реалистичних слика помоћу GAN-ова (*Generative Adversarial Networks*). Ови модели су ефикасни не само због прецизности, већ и због могућности интерпретације научених филтера и активација.

4.5 Савремене CNN архитектуре и еволуција

Развој конволутивних неуронских мрежа није стао на основним принципима, већ је током последње деценије значајно напредовао кроз појаву бројних архитектонских иновација. Свака нова архитектура представљала је одговор на специфичне изазове дубоког учења, као што су ефикасније учење, већа дубина мреже, бржа конвергенција и смањена потрошња меморије.

- **LeNet-5** (1998): Једна од првих CNN архитектура, дизајнирана за препознавање руком писаних цифара. Имала је пет слојева са конволуцијом и *subsampling*. Иако једноставна, показала је потенцијал CNN приступа.
- **AlexNet** (2012): Прва дубока CNN мрежа која је постигла велики успех на ImageNet такмичењу. Користила је ReLU активацију, dropout и GPU за паралелно тренирање, и значајно је допринела популаризацији дубоког учења.

- **VGGNet** (2014): Фокусирана на дубину мреже – користи мале 3x3 филтере и дубљу структуру са више конволуционих слојева. Омогућава боље препознавање хијерархијских шара.
- **GoogLeNet/Inception**: Увела тзв. *Inception module* који комбинује више филтера различитих величина паралелно унутар једног слоја. Тиме се обухватају информације на више скала.
- **ResNet** (2015): Решава проблем деградације перформанси код врло дубоких мрежа помоћу *residual connections*, које омогућавају директан пролаз сигнала (*skip connections*). ResNet архитектуре са 50+ слојева су постале стандард за многе задатке.
- **DenseNet**: Слична као ResNet, али сваки слој се повезује са свим претходним слојевима. Повећава искоришћеност информација и стабилизује ток градијената.

Ове архитектуре показале су да се перформансе CNN-а могу значајно побољшати не само повећањем сложености, већ и паметним дизајном топологије.

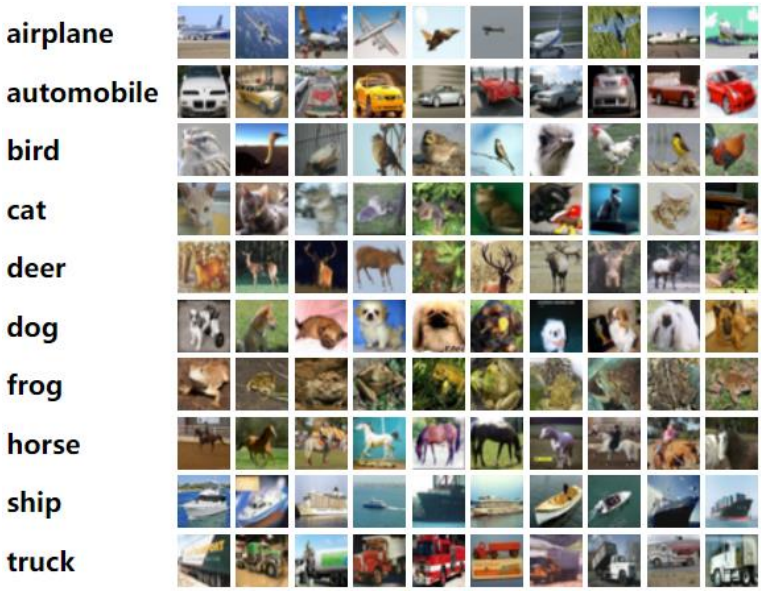
5. ИМПЛЕМЕНТАЦИЈА CNN-а

У овом поглављу приказано је како се уз помоћ савремених алата и техника из области дубоког учења може развити модел који аутоматски препознаје и класификује слике у више категорија.

Коришћен је CIFAR-10 скуп података, који садржи слике у боји распоређене у десет различитих класа. CIFAR-10 се састоји од укупно 60.000 слика, расподељених у 10 класа, Слика 21. Свака слика има димензије 32x32 пиксела и садржи три канала (RGB), што значи да је у боји. Слике су ниских резолуција, што омогућава брзо тренирање модела чак и на рачунарима без графичких картица. База је подељена на 50.000 слика за тренирање и 10.000 слика за тестирање. CIFAR-10 садржи следећих 10 класа, које представљају објекте из свакодневног живота:

1. Авион (*airplane*)
2. Аутомобил (*automobile*)
3. Птица (*bird*)
4. Мачка (*cat*)
5. Јелен (*deer*)
6. Пас (*dog*)
7. Жаба (*frog*)
8. Коњ (*horse*)
9. Брод (*ship*)
10. Камион (*truck*)

Свака класа садржи тачно 6.000 слика, што чини *dataset* потпуно балансираним.



Слика 21. CIFAR-10 класе

5.1 Најзначајније датотеке

За имплементацију модела коришћене су библиотеке *TensorFlow* и *Keras*, које омогућавају једноставно и ефикасно креирање неуронских мрежа, њихово тренирање и евалуацију. *TensorFlow* и *Keras* представљају две најзначајније и најшире коришћене технологије за изградњу неуронских мрежа и модела дубоког учења. Њихова интеграција омогућава једноставно, ефикасно и скалабилно креирање, тренирање и евалуацију комплексних модела.

TensorFlow

TensorFlow је *open-source* библиотека коју је развио *Google Brain* тим. Намењена је математичком моделирању, нумеричким прорачунима и развоју алгоритама дубоког учења. Основна структура у *TensorFlow*-у је тензор, који представља вишедимензионални низ података. Главне карактеристике *TensorFlow*-а су:

- Подршка за тренирање на CPU, GPU и TPU уређајима.
- Екстензивна подршка за визуализацију кроз *TensorBoard*.
- Могућност извођења модела на мобилним уређајима путем *TensorFlow Lite*.
- Дистрибуирано тренирање великих модела у реалном времену.

У образовне и експерименталне сврхе, *TensorFlow* омогућава брз и директан приступ познатим базама података (нпр. CIFAR-10, MNIST), чиме је олакшано учење и тестирање модела.

Keras

Keras је API (апликациони програмски интерфејс) који поједностављује рад са *TensorFlow*-ом. Од верзије *TensorFlow* 2.0, *Keras* је постао његов званични део под називом *tf.keras*. Главна улога *Keras*-а је да кориснику понуди јасан и интуитиван начин за дефинисање неуронских мрежа без потребе за писањем кодова ниског нивоа.

***Keras* омогућава:**

- Брзо формирање прототипа модела користећи *Sequential* и *Functional API*.
- Коришћење различитих слојева: *Dense, Conv2D, MaxPooling2D, Dropout*, итд.
- Једноставно компајлирање и тренирање модела путем *compile*(), *fit*() и *evaluate*() метода.
- Укључивање *callback* функција као што су *EarlyStopping, ModelCheckpoint* и *TensorBoard*.

Keras је идеалан за студенте и истраживаче јер нуди флексибилност, читљивост и лакоћу у експериментисању са различитим архитектурама неуронских мрежа.

Коришћењем *Keras*-а унутар *TensorFlow*-а (tf.keras), корисници добијају снагу *TensorFlow* машинерије уз једноставност *Keras* синтаксе. Модели могу да се тренирају уз високу прецизност, прате путем *TensorBoard*-а и извозе на различите платформе без додатних подешавања. *TensorFlow* и *Keras* су темељни алати у домену дубоког учења, омогућавајући креирање моћних и скалабилних модела на интуитиван начин. Њихова комбинација пружа одличну платформу како за истраживаче, тако и за индустријску примену. Захваљујући њиховој снази и приступачности, области попут препознавања слика, анализе језика и предикције података постају доступне свима, од ученика и студената до професионалних истраживача.

5.2 Структура и резултати имплементације

У оквиру имплементације развијене су две архитектуре:

1. Једноставна вештачка неуронска мрежа (NN) – користи само потпуно повезане слојеве (*Dense*) и представља базни модел.
2. Конволутивна неуронска мрежа (CNN) – користи конволутивне и поолинг слојеве, који су специјализовани за препознавање локалних шара у сликама.

Након тренирања оба модела на истим подацима, извршена је евалуација њихове тачности на тест скупу. Резултати су показали значајну разлику у перформансама:

- Једноставна неуронска мрежа (NN) остварила је тачност од око 46%, што је очекивано с обзиром на то да није способна да препозна просторне обрасце у слици.
- Конволутивна неуронска мрежа (CNN) постигла је тачност од преко 70%, захваљујући слојевима који ефикасно анализирају структуру слике и издвајају релевантне особине.

Ова разлика јасно показује супериорност конволутивне архитектуре у задацима визуелне класификације, нарочито када се ради о сликама са богатим локалним информацијама, као што су објекти у CIFAR-10 скупу.

Поред тренирања модела, имплементирана је и додатна функционалност којом се омогућава класификација слика које корисник унесе путем фолдера на свом *Google Drive* налогу. Тиме је обезбеђена практична демонстрација како се обучени модел може применити на нове, до тада невиђене податке. Такође, коришћен је *TensorBoard* за визуализацију процеса тренирања и праћење перформанси модела.

5.3 Комплетна анализа Python кода: CNN и класификација слика

1. Увоз библиотека и дефинисање ресурса

Овај део увози све потребне библиотеке:

- *TensorFlow* и *Keras* за рад са неуронским мрежама.

- *matplotlib.pyplot* за приказ слика.

- *numpy* за обраду података.

- *cv*2 за обраду слика.

- *PIL, math, pathlib* и *colab.files* као додатни алати.

```
import tensorflow as tf
import keras
import matplotlib.pyplot as plt
from tensorflow.keras import datasets, layers, models
import numpy as np
import cv2
from PIL import Image
import math
import pathlib
from google.colab import files
```

2. Путања до слика и класификационе ознаке

Дефинише се фолдер са сликама и листа имена класа које одговарају CIFAR-10 датасету.

```
path='/content/drive/MyDrive/slike'
klasifikacija=["avion","automobil","ptica","maca","jelen","pas","zaba","ko
њ","brod","kamionce"]
```

3. Учитавање и нормализација CIFAR-10 података

Учитава се CIFAR-10 *dataset* и пиксели се нормализују у опсегу 0-1.

```
(x_train,y_train),(x_test,y_test)=datasets.cifar10.load_data()
x_test=x_test/255
x_train=x_train/255
```

4. Дефинисање једноставне неуронске мреже (NN)

Модел се састоји од *Flatten* слоја, два *Dense* слоја са ReLU активацијом и једног излазног слоја са 10 неурона.

```
nn=models.Sequential([
            layers.Flatten(input_shape=(32,32,3)),
            layers.Dense(64,activation='relu'),
            layers.Dense(64,activation='relu'),
            layers.Dense(10,activation='sigmoid')
])
```

5. Компилација и тренирање обичне мреже

Модел се компајлира са *Adam* оптимизатором и тренира 25 епоха, затим се евалуира на тест скупу.

```
nn.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
nn.fit(x_train,y_train,epochs=25)
nn.evaluate(x_test,y_test)
```

6. Подешавање *TensorBoard*-а

Дефинише се *callback* који бележи податке о тренирању за приказ у *TensorBoard*-у.

```
tesnsor_board=tf.keras.callbacks.TensorBoard(log_dir="logs/fit",
histogram_freq=1)
```

7. Дефинисање CNN модела

Модел укључује два конволутивна слоја са *MaxPooling*-ом, *Flatten* слој и четири *Dense* слоја.

```
cnn=models.Sequential([

layers.Conv2D(filters=32,kernel_size=(3,3),activation='relu',input_shape=(
32,32,3)),
    layers.MaxPool2D(2,2),
    layers.Conv2D(filters=64,kernel_size=(3,3),activation='relu'),
    layers.MaxPool2D(2,2),
    layers.Flatten(),
    layers.Dense(64,activation='relu'),
    layers.Dense(64,activation='relu'),
    layers.Dense(64,activation='relu'),
    layers.Dense(64,activation='relu'),
    layers.Dense(10,activation='sigmoid')
])
```

8. Тренирање CNN-а уз *TensorBoard*

CNN се тренира 25 епоха уз праћење перформанси у *TensorBoard*-у.

```
cnn.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
cnn.fit(x_train,y_train,epochs=25,callbacks=[tesnsor_board])
cnn.evaluate(x_test,y_test)
```

9. Дефинисање функције за класификацију корисничких слика

Функција учитава слике из фолдера, нормализује их, користи модел за класификацију и приказује резултате.

```
def identifikuj(ime_putање, model, klasifikacije):
path=pathlib.Path(ime_putање)
pretrage=('*.jpg','*.jpeg','*.webp')
Test=[]
for m in pretrage:
       slike=list(path.glob(m))
       for i in slike:
```

```
        g=cv2.imread(str(i))
        g=cv2.resize(g,(32,32))
        Test.append(g)
Test1=np.array(Test)/255
print(Test1.shape)
 predikcije1=model.predict(Test1)
for l in range(len(Test1)):
        print(klasifikacije[np.argmax(predikcije1[l])])
        plt.figure()
        plt.imshow(Test1[l])
        plt.title(klasifikacije[np.argmax(predikcije1[l])])
return
```

10. Позивање функције класификације

Функција „идентификуј” се позива за класификацију слика у задатом фолдеру коришћењем тренираног CNN модела.

```
identifikuj('/content/drive/MyDrive/slike', cnn, klasifikacija)
```

6. ЗАКЉУЧАК

Конволутивне неуронске мреже (CNN) представљају један од најефикаснијих модела дубоког учења, посебно у задацима који укључују визуелне податке. Захваљујући својој архитектури, заснованој на локалној повезаности и слојевитој обради информација, CNN модели омогућавају аутоматско издвајање карактеристика из слика и других сложених улаза без потребе за ручним дефинисањем правила.

У оквиру овог рада обрађене су основе машинског учења, структура и функција неуронских мрежа, као и кључни појмови као што су функције грешке, активационе функције, алгоритми градијентног спуста и пропагације. Посебан акценат стављен је на рад конволутивних и *pooling* слојева, као и на примену техника попут регуларизације, трансфер учења и оптимизационих метода које побољшавају тачност и стабилност модела.

Практична имплементација CNN модела на CIFAR-10 скупу података показала је значајну надмоћ овог приступа у односу на класичну неуронску мрежу. Остварена је релативно висока тачност у класификацији слика, што потврђује применљивост ових мрежа у решавању реалних проблема, попут препознавања објеката и аутоматске анализе слика.

На основу анализе теоријских принципа и добијених резултата, може се закључити да конволутивне неуронске мреже имају велик потенцијал у даљем развоју вештачке интелигенције. Очекује се да ће будући напредак укључити бољу адаптацију на различите типове података, смањење потребе за великим количинама обележених примера и поједностављивање архитектура ради лакше примене у мобилним и уграђеним системима. Уз континуирани развој, CNN ће задржати своју водећу улогу у обради слике и другим применама где је потребно брзо и поуздано препознавање образаца.

7. ЛИТЕРАТУРА

1. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.
2. Crawford, K., & Calo, R. (2016). There is a blind spot in AI research. *Nature*, 538(7625), 311–313. https://doi.org/10.1038/538311a
3. Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, 55(10), 78–87. https://doi.org/10.1145/2347736.2347755
4. Fix, E., & Hodges, J. L. (1951). *Discriminatory analysis. Nonparametric discrimination: Consistency properties*. USAF School of Aviation Medicine.
5. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
6. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd ed.). Springer.
7. Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245), 255–260. https://doi.org/10.1126/science.aaa8415
8. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 1097–1105.
9. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444. https://doi.org/10.1038/nature14539
10. Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.
11. Müller, A. C., & Guido, S. (2017). *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media.
12. Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 386–408. https://doi.org/10.1037/h0042519
13. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088), 533–536. https://doi.org/10.1038/323533a0
14. Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.
15. Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 59(236), 433–460.
16. Vapnik, V., & Cortes, C. (1995). Support-vector networks. *Machine Learning*, 20(3), 273–297. https://doi.org/10.1007/BF00994018