Računarski fakultet

DIPLOMSKI RAD

Slika je vredna 16x16 reči: Vision Transformeri

Vanja Kovinić
RN 42/2020

Mentor:
dr Nemanja Ilić

Komisija:
dr Nemanja Ilić
dr Nevena Marić

Beograd, septembar 2024.

Sadržaj

Apstrakt

Ovaj diplomski rad istražuje **Vision Transformere** (**ViT**), nov pristup u oblasti računarskog vida koji koristi arhitekturu transformera prvobitno razvijenu za obradu prirodnog jezika. Prvi deo rada pruža detaljan pregled arhitekture transformera, uključujući ključne komponente kao što su ***self-attention*** **mehanizam** i **poziciono enkodovanje**, i diskutuje njihove svrhe i funkcionalnosti. Nakon toga, fokus se prebacuje na Vision Transformere, objašnjavajući kako se slike transformišu u **tokene** i obrađuju kroz **enkoder transformera** kako bi se primenili na rešavanje vizuelnih zadataka.

Rad zatim ulazi u praktične aspekte implementacije Vision Transformera, uključujući izbor i podešavanje **hiperparametara** za poboljšanje performansi. Izvršeno je i poređenje sa referentnim implementacijama, i predložen pristup za poboljšanje performansi. Prikazani su različiti eksperimenti, zajedno sa diskusijom njihovih rezultata, pružajući uvid u efikasnost i izazove povezane sa Vision Transformerima.

1 Uvod

"Pre otprilike 540 miliona godina, Zemlja je bila obavijena tamom. Ovo nije bilo zbog nedostatka svetlosti, već zato što organizmi još uvek nisu razvili sposobnost da vide. Iako je sunčeva svetlost mogla da prodre u okeane do dubine od 1.000 metara i hidrotermalni izvori na dnu mora isijavali svetlost u kojoj je život cvetao, nijedno oko nije se moglo naći u tim drevnim okeanima, nijedna retina, rožnjača ili sočivo. Sva svetlost i život nikada nisu viđeni. Koncept gledanja nije ni postojao tada i ova sposobnost nije ostvarena sve dok nije stvorena.

Iz nama nepoznatih razloga, trilobiti su se pojavili kao prva bića sposobna da spoznaju svetlost. Oni su prvi prepoznali da postoji nešto izvan njih samih, svet okružen višestrukim jedinkama. Smatra se da je rađanje vida pokrenulo kambrijsku eksploziju, period u kojem se veliki broj vrsta životinja pojavljuje u fosilnom zapisu. Vid je započeo kao pasivno iskustvo, jednostavno propuštanje svetlosti, ali je ubrzo postao aktivniji. Nervni sistem je počeo da evoluira, vid je prešao u uvid, gledanje je postalo razumevanje, razumevanje je dovelo do akcije, a sve to je dovelo do nastanka inteligencije.

Danas nismo više zadovoljni vizuelnom spoznajom koju nam je priroda dala. Radoznalost nas je navela da stvorimo mašine koje mogu da "vide" kao mi, pa čak i inteligentnije." - Li Fei-Fei [1]

1.1 Istorija i motivacija

1.1.1 Rani Razvoj u Računarskom Vidu

Koreni računarskog vida potiču iz ranih pokušaja da se razume i interpretira vizuelni podatak korišćenjem matematičkih modela i računara. U početku, istraživanja u oblasti računarskog vida fokusirala su se na jednostavne zadatke kao što su detekcija ivica, prepoznavanje objekata i osnovna obrada slika. Rane metode su se u velikoj meri oslanjale na ručnu izradu karakteristika (eng. ***features***) slike i algoritme dizajnirane da imitiraju osnovne aspekte ljudskog vida.

1.1.2 Uspon Dubokog Učenja

Značajan preokret u računarskom vidu dogodio se sa pojavom **dubokog učenja**. **Konvolucione Neuronske Mreže** (eng. **CNN**), koje su predstavili Yann LeCun i drugi [2] krajem 1980-ih i početkom 1990-ih, transformisale su ovu oblast uvođenjem automatskog ekstraktovanja karakteristika kroz slojeve koji se uče (eng. ***learnable features***). **CNN**-ovi su pokazali izuzetne performanse u različitim zadacima klasifikacije slika, omogućavajući računarima da nauče složene reprezentacije vizuelnih podataka. Ovo otkriće je kasnije propraćeno uspehom modela kao što su **AlexNet** [3], **VGGNet** [4] i **ResNet** [5], koji su postavili nove standarde u izazovima prepoznavanja slika.

1.1.3 Ograničenja CNN-ova

Uprkos svom uspehu, **CNN**-ovi imaju inherentna ograničenja [6] koja su motivisala potragu za novim pristupima. Jedan od značajnih nedostataka je njihova poteškoća u modelovanju udaljenih zavisnosti i globalnog konteksta unutar slike. **CNN**-ovi obično obrađuju slike kroz seriju lokalizovanih konvolucionih operacija, što može ograničiti njihovu sposobnost da razumeju odnose između udaljenih elemenata na slici.

1.1.4 Motivacija za uvođenje Vision Transformera

Pojava **Vision Transformera** (**ViT**) predstavlja odgovor na ova ograničenja. Inspirisani uspehom modela transformera u obradi prirodnog jezika (**NLP**), istraživači su pokušali da primene iste principe u računarskom vidu [7]. Transformeri koriste ***self-attention*** mehanizam za povezivanje globalnih zavisnosti, što ih čini pogodnim za zadatke koji zahtevaju razumevanje složenih odnosa unutar vizuelnih podataka.

Vision Transformeri rešavaju nekoliko izazova sa kojima se suočavaju **CNN**-ovi. Pretvaranjem slika u sekvence parčića (eng. ***image patches***) i primenom *self-attention* mehanizma preuzetim iz transformera, **ViT**-ovima mogu efikasnije modelovati globalni kontekst. Ovaj pristup omogućava **ViT**-ovima da postignu vrhunske performanse na različitim testovima klasifikacije slika i pokazuje njihov potencijal da unaprede oblast računarskog vida [7].

1.2 Konvolucione Neuronske Mreže (CNN)

1.2.1 Istorija CNN-ova

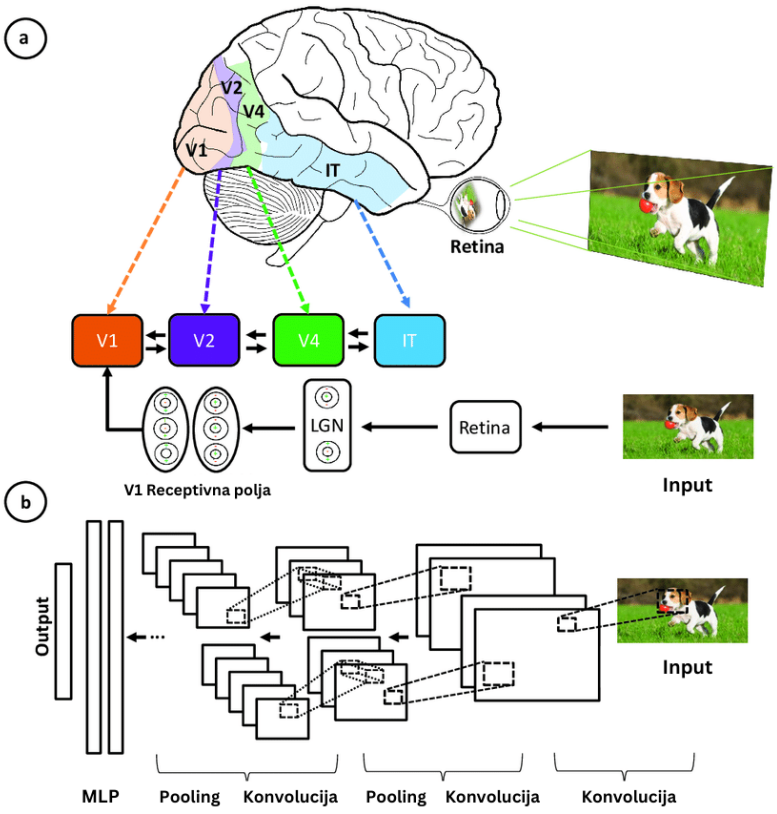
Konvolucione neuronske mreže su razvijene i prvi put korišćene oko 1980-ih. Tokom tog perioda, primarna primena **CNN**-ova bila je prepoznavanje rukom pisanih cifara, što je našlo praktičnu primenu u poštanskom sektoru za čitanje poštanskih i PIN kodova. Rani modeli **CNN**-ova, kao što je **LeNet** [8] koji je razvio Yann LeCun, pokazali su potencijal CNN-ova za zadatke prepoznavanja cifara [9].

Međutim, šira primena **CNN**-ova bila je ograničena značajnim izazovima. Duboki modeli učenja, uključujući **CNN**-ove, zahtevaju ogromne količine podataka za obuku i značajne računarske resurse, koji u to vreme nisu bili lako dostupni. Pored toga, ***backpropagation*** algoritam, koji je neophodan za obuku neuronskih mreža, bio je računarski skup. Ova ograničenja su suzila upotrebu **CNN**-ova uglavnom na poštanski sektor, i tehnologija nije uspela da stekne širu primenu u oblasti mašinskog učenja [10].

Oživljavanje **CNN**-ova došlo je 2012. godine, kada su Alex Krizhevsky, zajedno sa Ilyom Sutskeverom i Geoffreyjem Hintonom, prepoznali potencijal dubokog učenja sa višeslojnim neuronskim mrežama. Ovo oživljavanje je pokrenuto od strane nekoliko ključnih faktora: dostupnost velikih skupova podataka, kao što je ***ImageNet*** [11] skup sa milionima označenih slika, i značajna unapređenja u računarskim resursima, posebno **GPU**-ovima (eng. ***graphics processing unit***). Ovi razvojni događaji omogućili su istraživačima da prevaziđu prethodna ograničenja i u potpunosti iskoriste mogućnosti konvolucionih neuronskih mreža [10].

1.2.2 Ljudski vizuelni sistem kao inspiracija

Arhitektura konvolucionih neuronskih mreža je analogna načinu na koji su neuroni u ljudskom mozgu povezani i inspirisana je organizacijom **vizuelnog korteksa** (*Slika 1*). Pojedinačni neuroni reaguju na stimuluse samo u ograničenom delu vizuelnog polja poznatom kao **receptivno polje** (eng. ***receptive field***). Ova polja se preklapaju kako bi se pokrilo čitavo vizuelno polje.



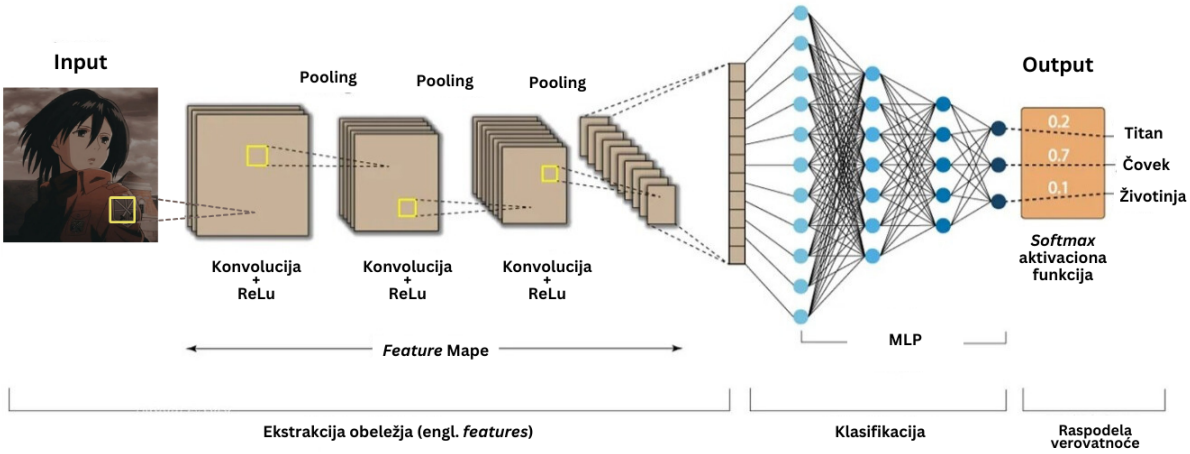
Slika 1: Poređenje između organizacije vizuelnog korteksa i **CNN** arhitekture [12]

Prema Keitu [13], glavne sličnosti između organizacije vizuelnog korteksa i arhitekture **CNN**-ova su:

- **Hijerarhijska arhitektura**: I **CNN** i vizuelni korteks imaju hijerarhijsku strukturu, gde se jednostavni oblici izvlače u početnim slojevima, a složeniji oblici se grade u dubljim slojevima. Ovo omogućava kompleksniju reprezentaciju vizuelnog inputa.
- **Lokalna povezanost**: Neuroni u vizuelnom korteksu su povezani samo sa lokalnim regionom inputa, a ne sa celim vizuelnim poljem. Slično tome, neuroni u sloju **CNN**-a su povezani samo sa lokalnim regionom inputa putem **konvolucione operacije**. Ova lokalna povezanost omogućava efikasnost.
- **Translaciona invarijantnost**: Neuroni vizuelnog korteksa mogu detektovati karakteristike bez obzira na njihovu lokaciju u vizuelnom polju. ***Pooling*** slojevi u **CNN**-u pružaju određeni stepen **translacione invarijantnosti**.
- **Višestruke *feature* mape**: Na svakoj fazi vizuelne obrade, izvlače se različite ***feature*** mape. **CNN**-ovi ovo imitiraju putem višestrukih jezgara (eng. ***kernels***) koji detektuju različite karakteristike u svakom konvolucionom sloju.
- **Nelinearnost**: Neuroni u vizuelnom korteksu pokazuju osobine nelinearnosti. **CNN**-ovi postižu nelinearnost putem **aktivacionih funkcija**, koje se primenjuju nakon svake konvolucije.

1.2.3 Arhitektura CNN-ova

Arhitektura **CNN**-a biće prikazana na primeru klasifikacionog problema[1], gde je cilj modela da klasifikuje slike u jednu od P klasa, kao što je prikazano na *Slici 2.*



Slika 2: Primer arhitekture **CNN**-a za klasifikaciju slika

Sastavni deo **CNN**-a su:

- **Konvolucioni slojevi**
- ***Pooling* slojevi**
- **Višeslojni perceptron (eng. *Multi-Layer Perceptron* - MLP)**

[1]Gradivni elementi CNN-a su identični, bez obzira na to da li je u pitanju problem regresione ili neke druge prirode; jedina razlika leži u MLP glavi.

Konvolucioni slojevi

1. Konvoluciona operacija

U **konvolucionom sloju**, osnovna operacija je **konvolucija** ulazne slike sa **kernelom** (poznatim i kao **filter**). Cilj ove operacije je detektovanje karakteristika kao što su ivice, teksture ili šare u ulaznim podacima.

Neka je ulazna slika predstavljena kao 2D matrica I sa dimenzijama $H \times W$, gde je H visina, a W širina slike. Neka je K 2D kernel sa dimenzijama $f \times f$, gde je f veličina kernela.

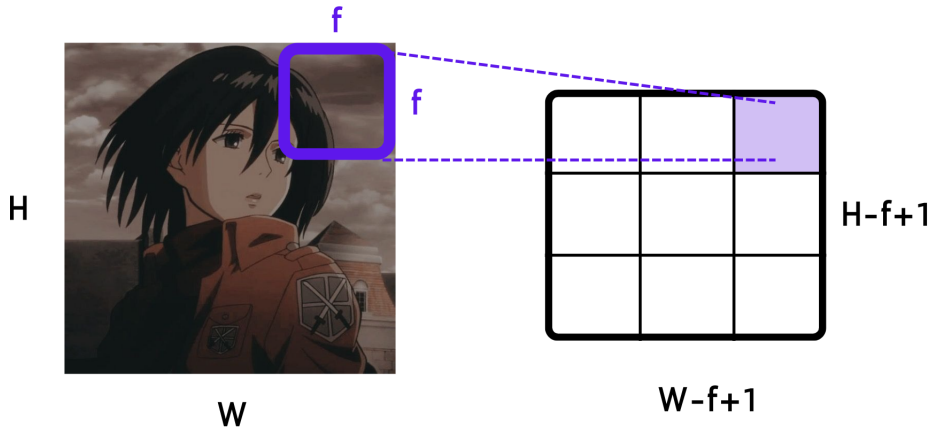
Operacija konvolucije se može matematički izraziti kao:

$$S(i,j) = \sum_{m=0}^{f-1} \sum_{n=0}^{f-1} I(i+m, j+n) \cdot K(m,n)$$

gde $S(i,j)$ predstavlja vrednost izlazne ***feature*** mape na poziciji (i,j). Ovde (i,j) označava poziciju na ulaznoj slici gde se primenjuje kernel [14].

2. Izlaz konvolucione operacije

Rezultat primene kernela na ulaznu sliku je *feature* mapa F sa dimenzijama $(H - f + 1) \times (W - f + 1)$, pod pretpostavkom da se ne koristi *padding* (*Slika 3*). Veličina izlazne *feature* mape je smanjena u odnosu na ulaznu sliku zbog klizne operacije kernela preko ulazne slike.



Slika 3: Dimenzije ulazne slike i *feature* mape u jednom konvolucionom sloju

3. Popunjavanje (eng. Padding)

Popunjavanje se koristi za kontrolu prostornih dimenzija izlazne *feature* mape [15]. Popunjavanje predstavlja dodavanje piksele oko ivica ulazne slike. Neka je p broj piksela koji se dodaju. Popunjena ulazna slika I' ima dimenzije $(H + 2p) \times (W + 2p)$.

Operacija konvolucije sa popunjavanjem može se izraziti kao:

$$S(i,j) = \sum_{m=0}^{f-1} \sum_{n=0}^{f-1} I'(i+m, j+n) \cdot K(m,n).$$

4. Korak (eng. Stride)

Korak određuje koliko piksela se filter pomera pri svakom koraku tokom konvolucije [16]. Neka je s vrednost koraka. Korak utiče na dimenzije izlazne *feature* mape. Sa korakom s, izlazna *feature* mapa F ima dimenzije:

$$H_{\text{out}} = \frac{H - f + 2p}{s} + 1$$

$$W_{\text{out}} = \frac{W - f + 2p}{s} + 1$$

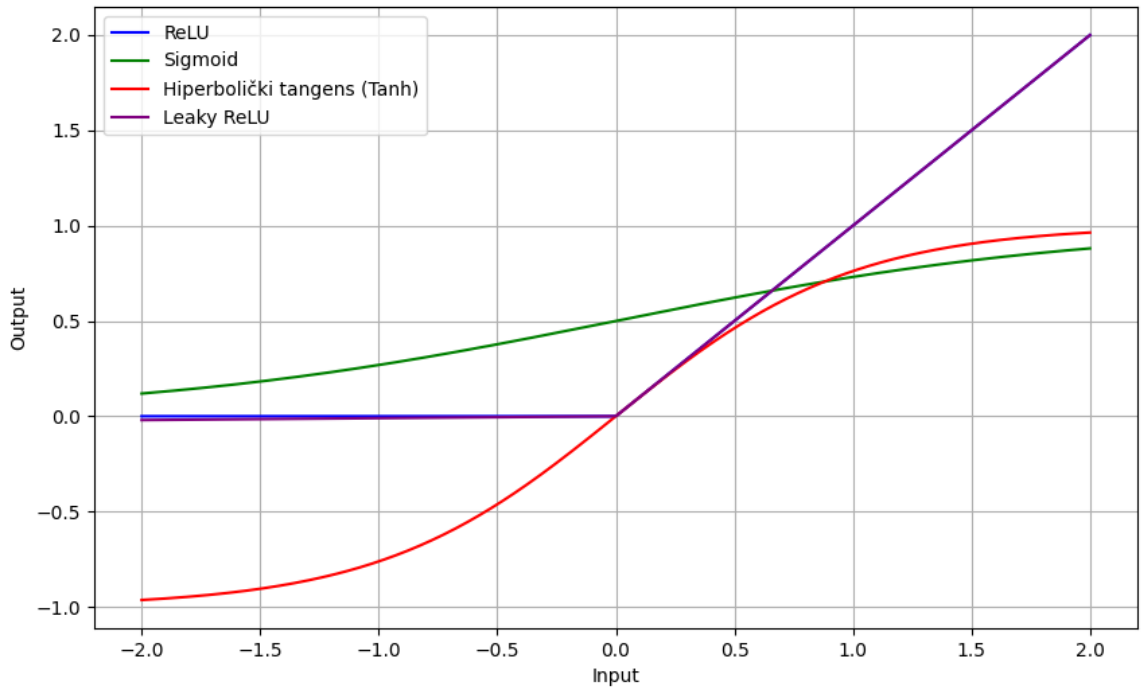
gde su H_{out} i W_{out} visina i širina izlazne *feature* mape, respektivno.

5. Aktivaciona Funkcija

Nakon konvolucione operacije, aktivaciona funkcija se primenjuje element po element da bi se uvela nelinearnost u model. Aktivaciona funkcija koja se najčešće koristi je **ReLU** (*Rectified Linear Unit*), definisana kao:

$$\text{ReLU}(x) = \max(0, x)$$

gde je x ulaz u aktivacionu funkciju. Neke od najpopularnijih aktivacionih funkcija se mogu videti na *Slici 4*.



Slika 4: Grafik najčešće korišćenih aktivacionih funkcija

Pooling slojevi

Pooling slojevi su još jedna ključna komponenta **CNN**-ova koja vrši operaciju uzorkovanja po određenoj strategiji kako bi se smanjile prostorne dimenzije ulazne *feature* mape, čime se smanjuje računarska složenost i sprečava prekomerno prilagođavanje (eng. *overfitting*). Najčešće korišćene operacije pooling-a su ***max pooling*** i ***average pooling*** [17].

Pooling operacija

Pooling operacije se primenjuju na svaku *feature* mapu nezavisno. Ulazna *feature* mapa F ima dimenzije $H \times W$, gde je H visina, a W širina. *Pooling* operacija pomera prozor veličine $f \times f$ preko *feature* mape, sa korakom s.

Max Pooling

Kod *max pooling* operacije, izlazna vrednost za svaki prozor je maksimalna vrednost unutar tog prozora. Matematički, za *feature* mapu F i *pooling* prozor veličine $f \times f$ sa korakom s, *max pooling* operacija se može izraziti kao:

$$M(i,j) = \max_{0 \leq m < f, 0 \leq n < f} F(s \cdot i + m, s \cdot j + n)$$

gde $M(i,j)$ predstavlja vrednost izlazne *feature* mape na poziciji (i,j).

Average Pooling

Kod *average pooling* operacije, izlazna vrednost za svaki prozor je prosečna vrednost unutar tog prozora. Matematički, za *feature* mapu F i *pooling* prozor veličine $f \times f$ sa korakom s, *average pooling* operacija se može izraziti kao:

$$A(i,j) = \frac{1}{f^2} \sum_{m=0}^{f-1} \sum_{n=0}^{f-1} F(s \cdot i + m, s \cdot j + n)$$

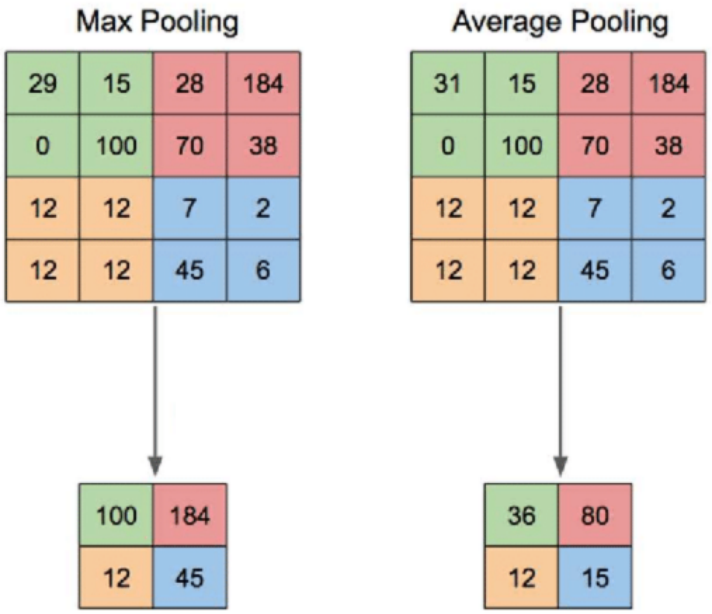
gde $A(i,j)$ predstavlja vrednost izlazne mape karakteristika na poziciji (i,j).

Izlaz Pooling-a

Dimenzije izlazne *feature* mape zavise od veličine prozora za pooling f i koraka s. Za datu ulaznu *feature* mapu F dimenzija $H \times W$, izlazna *feature* mapa O (bilo M za *max pooling* ili A za *average pooling*) ima dimenzije:

$$H_{\text{out}} = \frac{H - f}{s} + 1$$
$$W_{\text{out}} = \frac{W - f}{s} + 1$$

gde su H_{out} i W_{out} visina i širina izlazne *feature* mape, respektivno (ilustracija *pooling* operacije na *Slici 5*).



Slika 5: Primer izvođenja *max pooling* i *average pooling* operacije [17]

Višeslojni Perceptron (MLP)

Perceptron

Perceptron prima više ulaznih signala, primenjuje težine na njih, sabira ih i prosleđuje rezultat kroz aktivacionu funkciju kako bi proizveo izlaz (*Slika 6*).

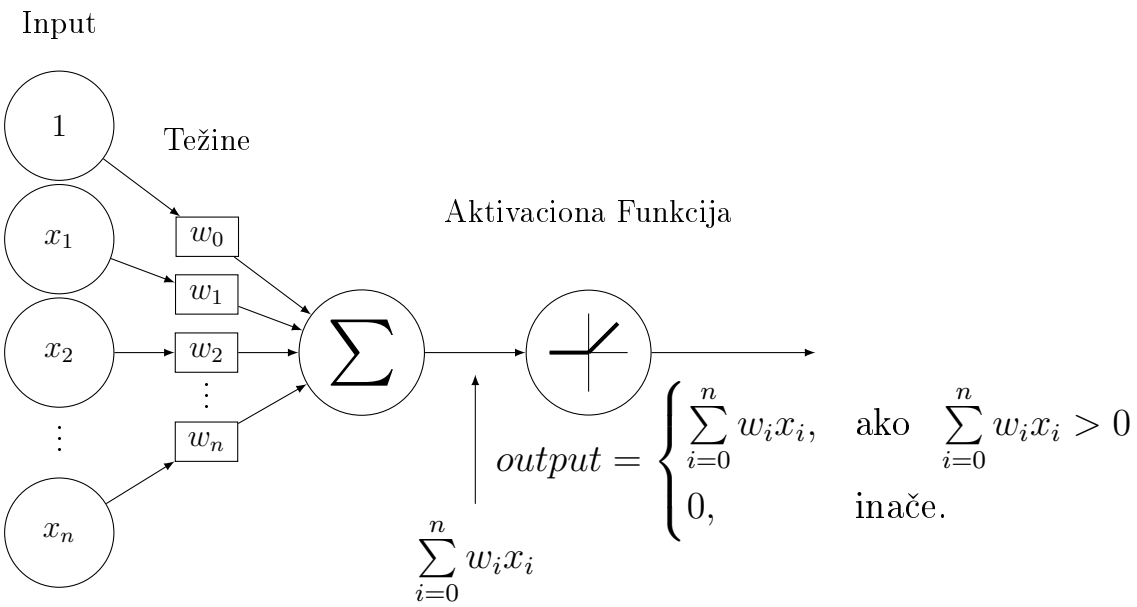
Za dat ulazni vektor $\mathbf{x} = [x_0, x_2, \ldots, x_n]$ i odgovarajuće težine $\mathbf{w} = [w_0, w_2, \ldots, w_n]$, perceptron računa ponderisanu sumu na sledeći način:

$$z = \sum_{i=0}^{n} w_i x_i$$

gde je $x_0 = 1$, pa je $x_0 w_0$ slobodan član (često se još naziva i pomeraj, a obeležava kao b, eng. *bias*).

Izlaz y se zatim dobija primenom aktivacione funkcije $f(z)$. U našem primeru smo za aktivacionu funkciju koristili **ReLU** funkciju, koja se definiše na sledeći način:

$$y = \begin{cases} z, & \text{ako } z > 0 \\ 0, & \text{inače.} \end{cases}$$



Slika 6: Grafički prikaz perceptrona

Višeslojni Perceptron (MLP)

MLP je potpuno povezana veštačka neuronska mreža koja se sastoji od više slojeva perceptrona, obično uključujući ulazni sloj, jedan ili više skrivenih slojeva i izlazni sloj [18].

Za **MLP** sa L slojeva, ulaz u mrežu je $\mathbf{x} \in \mathbb{R}^n$. Izlaz svakog sloja l se računa kao:

$$\mathbf{h}^{(l)} = f(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)})$$

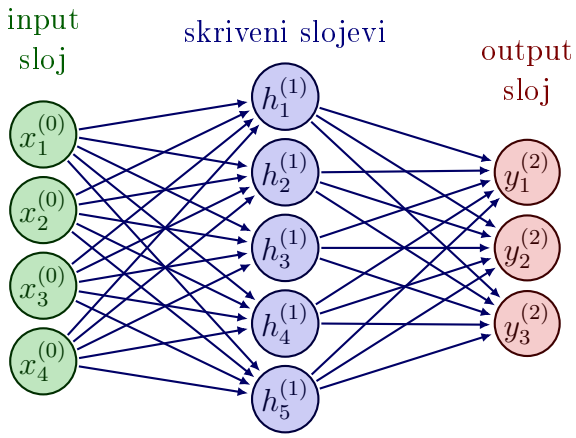
gde:

- $\mathbf{h}^{(0)} = \mathbf{x}$ je ulazni vektor,
- $\mathbf{W}^{(l)}$ i $\mathbf{b}^{(l)}$ su matrica težina i vektor pomeraja za sloj l,
- f je aktivaciona funkcija (obično $ReLU$, $sigmoid$ ili $tanh$).

Finalni sloj često koristi *softmax* funkciju za klasifikacione zadatke, koja pretvara izlaz u distribuciju verovatnoća za date klase:

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{P} e^{z_j}}$$

gde je $\mathbf{z}$ ulaz u softmax funkciju, a P je broj klasa.



Slika 7: Grafički prikaz **MLP**-a sa 4 ulazna, 5 skrivenih i 3 izlazna čvora

2 Transformeri

2.1 Istorija i razvoj

Model **Transformera**, predstavljen u seminalnom radu *"Attention is All You Need"* [19] od Vaswani-ja i saradnika 2017. godine, označio je značajan napredak u oblasti obrade prirodnog jezika (**NLP**). Pre toga, modeli kao što su Rekurentne Neuronske Mreže (**RNN**) [20] i *Long Short-Term Memory* Mreže (**LSTM**) [21] bili su dominantne arhitekture za *sequence-to-sequence* zadatke. Međutim, ovi modeli su imali ograničenja, posebno sa udaljenim zavisnostima i paralelizacijom.

Transformeri su revolucionisali **NLP** uvođenjem nove arhitekture koja se u potpunosti zasniva na ***self-attention*** mehanizmu, eliminišući potrebu za rekurentnim slojevima. Ova inovacija je omogućila efikasnije treniranje i sposobnost da se bolje modeluju veze između udaljenih reči u sekvenci. Uvođenje transformera dovelo je do dramatičnog poboljšanja performansi na različitim **NLP** zadacima, kao što su mašinsko prevođenje, sažimanje teksta i odgovaranje na pitanja.

Njihov uspeh brzo je postao evidentan razvojem moćnih modela koji su ih koristili kao osnovu. Jedna od prvih značajnih primena bila je u mašinskom prevođenju, gde je transformer nadmašio prethodne najbolje modele na referentnim skupovima podataka. Ovaj uspeh je dodatno pojačan stvaranjem modela kao što su **BERT** *(Bidirectional Encoder Representations from Transformers)* [22] i **GPT** *(Generative Pre-trained Transformer)* [23], koji su postavili nove standarde za različite **NLP** zadatke.

2.2 Motivacija za transformer arhitekturu

Pre pojave transformera, Rekurentne Neuronske Mreže (**RNN**) i *Long Short-Term Memory* Mreže (**LSTM**) bile su primarni modeli korišćeni za zadatke sa sekvencijalnim podacima u **NLP**-u. Iako su ovi modeli bili donekle efikasni, imali su ograničenja naročito kod zadataka koji uključuju udaljene zavisnosti [24].

2.2.1 Rekurentne Neuronske Mreže (RNN)

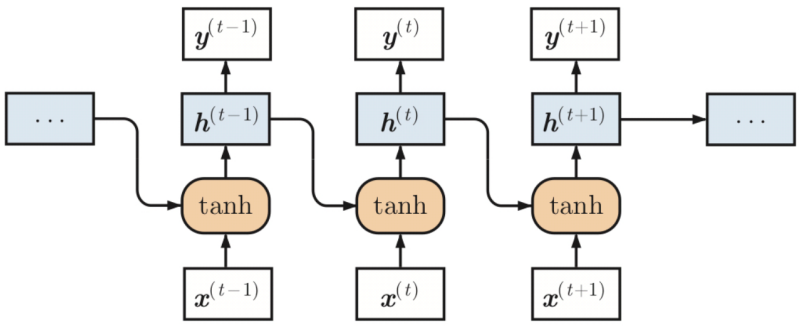
RNN-ovi obrađuju sekvence podataka održavanjem skrivenog stanja (eng. ***hidden state***) koje čuva informacije o prethodnim elementima u sekvenci (*Slika 8*). Skriveno stanje u vremenskom koraku t, označeno kao h_t, se računa kao:

$$h_t = f(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

gde:

- x_t je ulaz u vremenskom koraku t,
- W_{hh} i W_{xh} su matrice težina,
- b_h je pomeraj,
- f je aktivaciona funkcija (npr. tanh).

RNN-ovi pate od problema nestajućih i eksplodirajućih gradijenata (eng. ***vanishing and exploding gradients***), što otežava efikasno učenje udaljenih zavisnosti. Gradijenti *loss* funkcije ili opadaju ili rastu eksponencijalno tokom propagacije unazad (***backpropagation***), što dovodi do slabog učenja udaljenih zavisnosti [25].



Slika 8: Prikaz rekurentne neuronske mreže [26]

2.2.2 Long Short-Term Memory Mreže (LSTM)

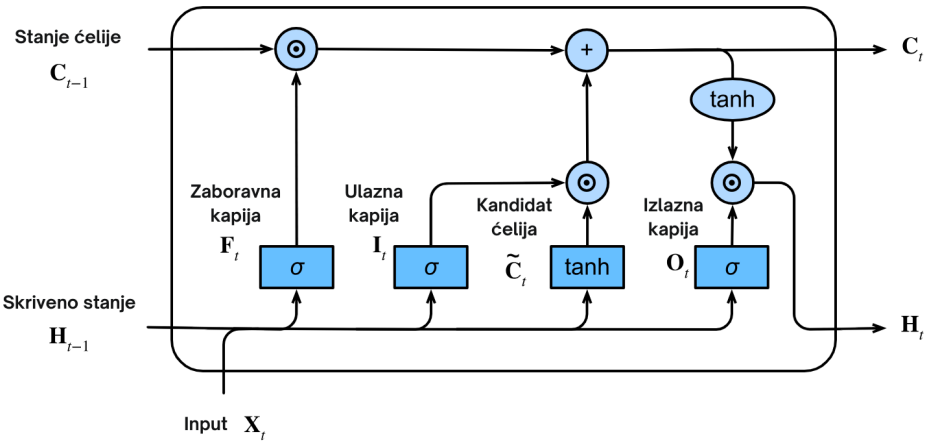
LSTM-ovi su uvedeni da bi rešili neka od ograničenja **RNN**-ova [27]. Uključuju memorijske ćelije i mehanizme za kontrolu protoka informacija, omogućavajući im da efikasnije modeluju udaljene zavisnosti. Ključne komponente **LSTM**-a su ćelije koje se sastoje od: ulazne kapije (i_t), zaboravne kapije (f_t), izlazne kapije (o_t) i stanja ćelije (c_t), detaljnije na *Slici 9*. Stanje ćelije se ažurira kao:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

gde je $\odot$ Adamarov proizvod, a $\tilde{c}_t$ kandidat za stanje ćelije, obično računat kao:

$$\tilde{c}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c).$$

Iako **LSTM**-ovi ublažavaju neke probleme **RNN**-ova, i dalje se suočavaju sa izazovima u paralelizaciji i računskoj efikasnosti. Sekvencijalna priroda njihove obrade znači da ne mogu efikasno koristiti paralelno izračunavanje, što ograničava njihovu skalabilnost.



Slika 9: Prikaz arhitekture **LSTM** modela [28]

2.2.3 Prednost transformera

Transformeri rešavaju ova ograničenja koristeći potpuno drugačiji mehanizam poznat kao ***self-attention*** mehanizam, koji omogućava efikasnu paralelizaciju i poboljšano rukovanje udaljenim zavisnostima. *Self-attention* mehanizam računa *attention* rezultate za svaki par elemenata u ulaznoj sekvenci, modelujući zavisnosti bez obzira na njihovu udaljenost. Ovaj mehanizam omogućava transformer modelu da procesira sve elemente sekvence istovremeno, omogućavajući efikasnu paralelizaciju.

Pored toga, korišćenje ***multi-head attention*** mehanizma dodatno poboljšava sposobnost mreže da modeluje različite aspekte odnosa između elemenata u sekvenci.

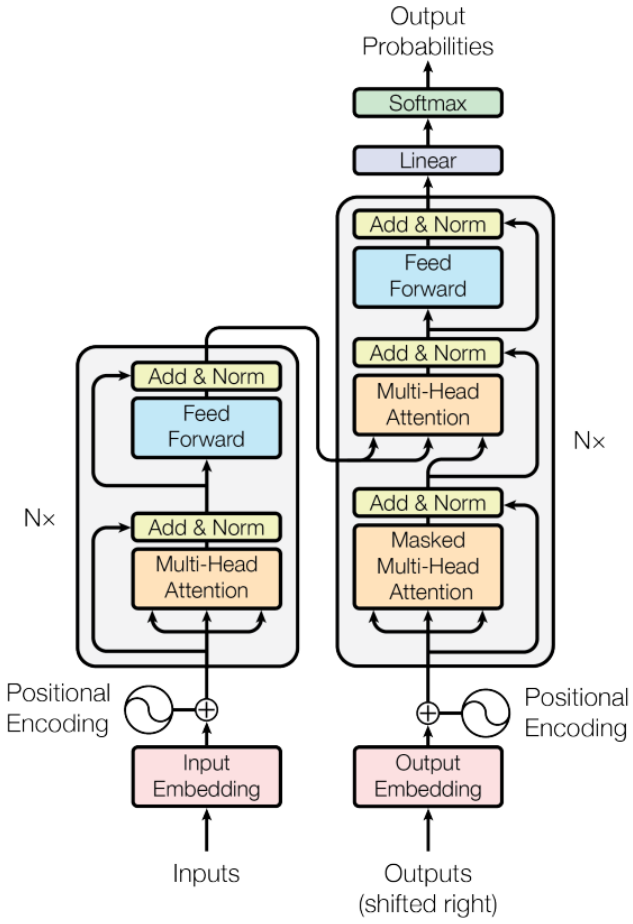
Korišćenjem *self-attention* i *multi-head attention* mehanizama, transformeri prevazilaze ograničenja **RNN**-ova i **LSTM**-ova, pružajući značajna poboljšanja u performansama i efikasnosti za širok spektar **NLP** zadataka [19].

2.3 Arhitektura transformera

Arhitektura transformera sastoji se od dva glavna dela (*Slika 10*):

- **Enkoder**
- **Dekoder**

Obe komponente su dizajnirane da rade zajedno u zadacima poput mašinskog prevođenja, gde enkoder obrađuje ulaznu sekvencu, a dekoder generiše odgovarajuću izlaznu sekvencu.



Slika 10: Transformer arhitektura [19]

2.3.1 Enkoder

Enkoder je odgovoran za obradu ulazne sekvence i ekstrahovanje njenih reprezentacija [29].

Input Embedding

Pre nego što se bilo koja ulazna sekvenca može obraditi od strane transformera, ona mora biti konvertovana u neprekidnu vektorsku reprezentaciju. Ovo se postiže kroz *embedding* sloj , koji mapira svaki token[2] ulazne sekvence u vektor fiksne veličine, označen sa d_{model}, kao što je prikazano na *Slici 11*.

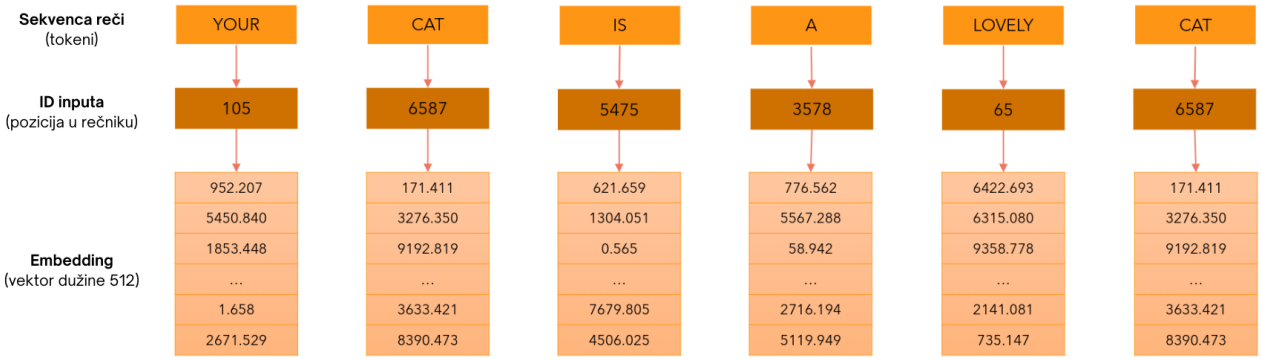
[2]Token predstavlja osnovnu jedinicu informacije, koja može biti reč, deo reči ili simbol, a koristi se kao ulazni podatak u enkoder.

Za datu ulaznu sekvencu tokena $\{x_1, x_2, \ldots, x_{\text{seq}}\}$, svaki token x_i se mapira u *embedding* vektor $E(x_i)$ veličine d_{model}. Ova mapiranja se uče tokom procesa treniranja i obično se predstavljaju kao:

$$E : \{x_1, x_2, \ldots, x_{\text{seq}}\} \to \mathbb{R}^{d_{\text{model}}}.$$

Tako se ulazna sekvenca $\{x_1, x_2, \ldots, x_{\text{seq}}\}$ transformiše u *embedding* matricu $X \in \mathbb{R}^{\text{seq} \times d_{\text{model}}}$, gde svaki red odgovara embedovanju određenog tokena u sekvenci:

$$X = \begin{bmatrix} E(x_1) \\ E(x_2) \\ \vdots \\ E(x_{\text{seq}}) \end{bmatrix}$$



Slika 11: Embedovanje tokena u vektorski prostor [30]

Pozicionо enkodovanje (eng. positional encoding)

Da bi model bio sposoban da prepozna redosled tokena u sekvenci, pozicioni vektori se dodaju na *embedding* vektore. Ovi vektori unose informaciju o poziciji svakog tokena, omogućavajući modelu da razlikuje tokene na osnovu njihove pozicije u sekvenci [19].

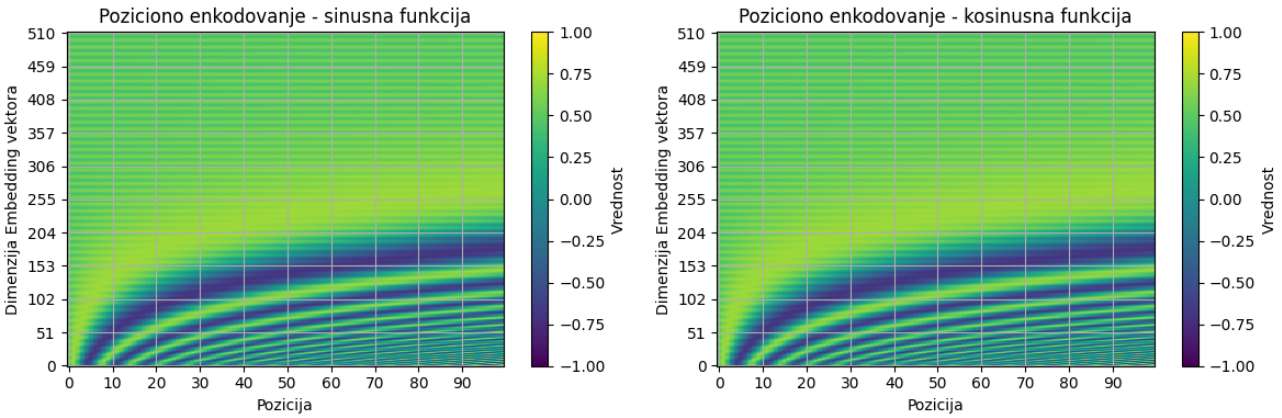
Pozicioni vektori mogu biti naučeni, kao što se uče i *embedding* vektori, ili mogu biti fiksni, zasnovani na unapred definisanim funkcijama. Autori transformera odlučili su da koriste fiksne vektore, koji su davali gotovo iste rezultate kao i naučeni, dok su pritom bili jednostavniji i lakši za interpretaciju.

Fiksni pozicioni vektori (*Slika 12*) za svaku poziciju *pos* u sekvenci definisani su kao:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

gde:

- *pos* je pozicija tokena u sekvenci.
- i je indeks dimenzije (od 0 do $d_{\text{model}}/2 - 1$).



Slika 12: Grafik pozicionog enkodovanja za sinusnu i kosinusnu funkciju

Korišćenje **sinusnih** i **kosinusnih** funkcija osigurava da su pozicioni vektori neprekidni i glatki, što pomaže u očuvanju relativnih pozicionih informacija.

Pored toga, ove funkcije imaju periodične osobine koje omogućavaju modelu da generalizuje na sekvence različitih dužina, jer slične pozicije proizvode slične pozicione vektore čak i u dugačkim sekvencama.

Ovi pozicioni vektori se dodaju na *embedding* vektore:

$$X' = X + PE$$

gde je X' konačni ulaz u enkoder, koji kombinuje *embedding* vektor sa pozicionim vektorom. Ova kombinacija omogućava transformeru da iskoristi i sadržaj i redosled tokena u sekvenci, što omogućava efikasnije učenje i predikciju [19].

Self-Attention mehanizam

Self-attention **mehanizam** [19] omogućava modelu da odredi značaj nekog tokena u ulaznoj sekvenci u odnosu na ostale ulazne tokene. Na taj način model može da modeluje zavisnosti između tokena, bez obzira na njihovu udaljenost u sekvenci, što predstavlja značajnu prednost u odnosu na tradicionalne rekurentne neuronske mreže (**RNN**).

Težine, odnosno *attention* vrednosti, predstavljaju koliko fokusa model treba da usmeri na ostale tokene prilikom obrade određenog tokena [29]. Ove vrednosti se izračunavaju koristeći tri ključne komponente izvedene iz ulaznih *embedding* vektora:

- **Upit (Q, Query):** Predstavlja token koji trenutno obrađujemo,
- **Ključ (K, Key):** Predstavlja tokene u sekvenci sa kojima se upoređuje trenutni token,
- **Vrednost (V, Value):** Predstavlja stvarne informacije tokena koje će se koristiti kasnije u množenju.

Za svaki token u sekvenci, računamo vektore **Upita**, **Ključa** i **Vrednosti**:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

gde su:

- X matrica ulaznih *embedding* vektora dimenzija seq $\times\, d_{\text{model}}$,
- W^Q, W^K, W^V matrice težina dimenzija $d_{\text{model}} \times d_k$, $d_{\text{model}} \times d_k$, i $d_{\text{model}} \times d_v$,
- d_k i d_v dimenzije vektora **Upita**/**Ključa** i **Vrednosti**, često su $d_k = d_v = d_{\text{model}}/h$, gde je h broj *self-attention* glava (o čemu će biti reči u narednom odeljku).

Attention vrednosti za svaki par tokena se izračunavaju kao skalarni proizvod **Upita** trenutnog tokena sa **Ključem** drugog tokena, skaliran kvadratnim korenom dimenzije d_k, nakon čega sledi *softmax* funkcija koja normalizuje rezultate (*Slika 13*):

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V.$$

Skaliranje sa $\sqrt{d_k}$ sprečava da skalarni proizvod postane prevelik, što bi moglo dovesti do vrlo malih gradijenata i sporog treniranja.

Slika 13: Prikaz *self-attention* mehanizma [30]

Multi-Head Attention

Kako bi se modelovali različiti tipovi odnosa između tokena, Transformer koristi *multi-head attention* mehanizam. Umesto da se izračuna jedan skup vektora **Upita**, **Ključa** i **Vrednosti**, ulaz se projektuje u više podprostora (glava), od kojih svaka ima svoj skup matrica težina (*Slika 14*). Ovo omogućava modelu da nauči različite aspekte odnosa između tokena [29].

Za svaku glavu i:

$$Q_i = XW_i^Q, \quad K_i = XW_i^K, \quad V_i = XW_i^V.$$

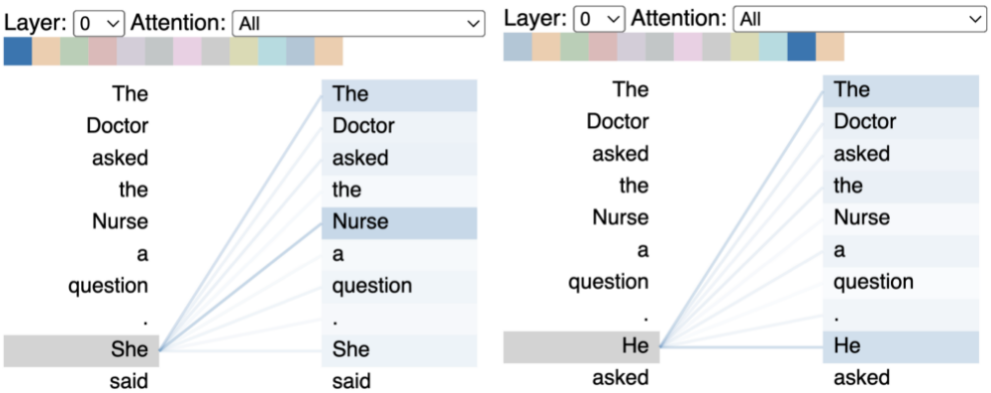
Svaka glava izračunava svoj izlaz operatora pažnje:

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i).$$

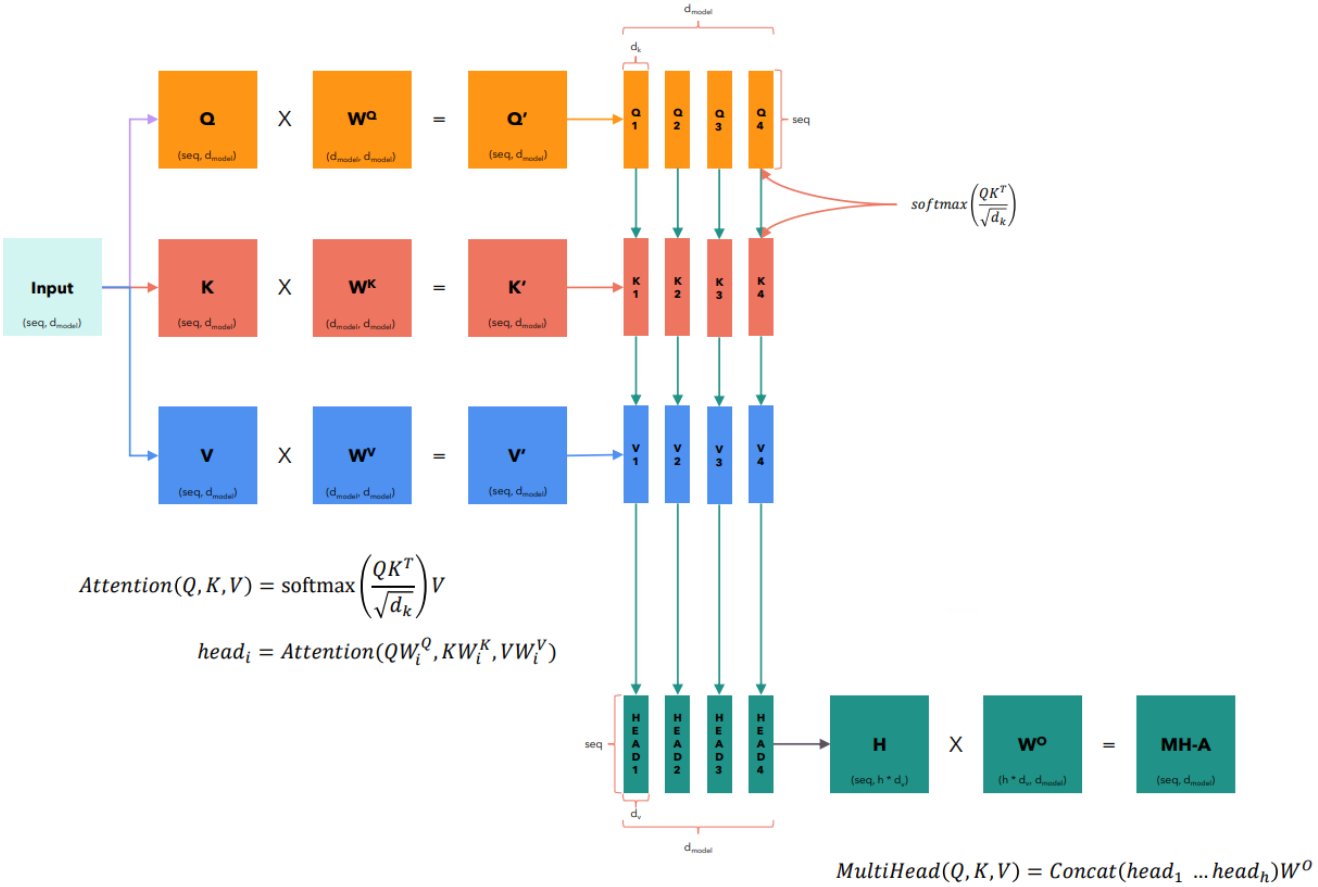
Izlazi svih glava se zatim konkateniraju i projektuju nazad na originalnu dimenziju d_{model}:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

gde je W^O matrica težina dimenzija $hd_v \times d_{\text{model}}$ (*Slika 15*).



Slika 14: Vizuelizacija jedne glave *multi-head attention* mehanizma [31]

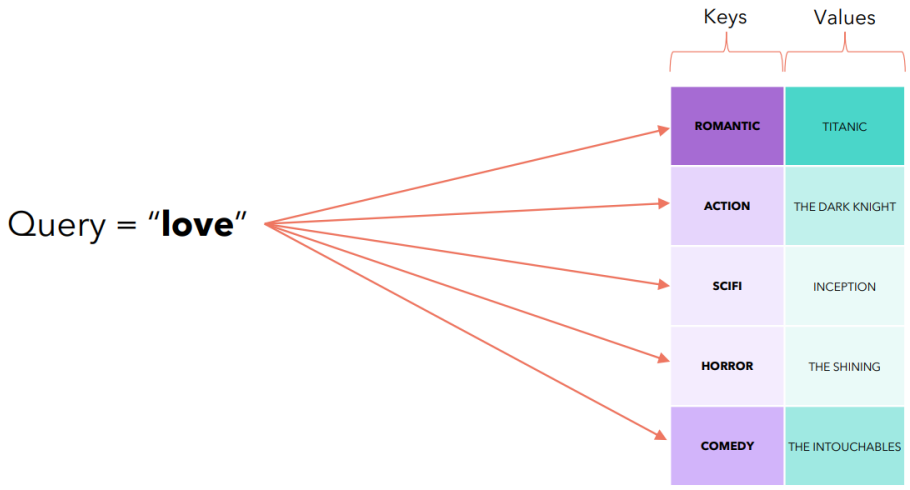


Slika 15: Grafik *multi-head attention* mehanizma [30]

Intuicija i Uticaj

Notacija K, Q i V inspirisana je konceptima ključa (key), upita (query) i vrednosti (value) iz sistema baza podataka (*Slika 16*).

Self-attention mehanizam omogućava transformeru da razume kontekst reči u sekvenci gledajući druge reči u sekvenci, bez obzira na njihov položaj [29]. Ovo je ključno za modelovanje udaljenih zavisnosti, čineći transformere posebno efikasnim u zadacima kao što su jezičko modelovanje, prevođenje i, kao što je to slučaj kod Vision Transformera, klasifikacija slika.



Slika 16: Koncepti ključa, upita i vrednosti dolaze iz baza podataka [30]

Rezidualne Konekcije i Normalizacija Slojeva

Rezidualne konekcije i normalizacija slojeva su ključne tehnike koje poboljšavaju stabilnost i performanse tokom obuke.

Rezidualne konekcije (*skip* konekcije) omogućavaju da se ulaz sloja direktno doda njegovom izlazu, što pomaže u rešavanju problema nestajanja gradijenta. Ovo olakšava obuku dubljih mreža omogućavajući da gradijenti lakše teku kroz mrežu.

Formalno:

$$\text{Output} = \text{Layer}(x) + x$$

gde:

- x je ulaz u sloj,
- $\text{Layer}(x)$ je izlaz transformacije sloja.

Ovo dodavanje rezidualne konekcije omogućava modelu da očuva originalne informacije.

Normalizacija slojeva stabilizuje i ubrzava obuku normalizovanjem aktivacija duž svih dimenzija svakog uzorka ponaosob. Razlikuje se od *batch* normalizacije po tome što normalizuje svaki pojedinačni uzorak umesto duž cele serije [32].

Formalno:

$$\text{LayerNorm}(x) = \frac{x - \mu}{\sigma} \cdot \gamma + \beta$$

gde:

- x je ulazni vektor,
- μ i σ su srednja vrednost i standardna devijacija x,
- γ i β su parametri za skaliranje i pomeranje koji se uče.

U transformeru, normalizacija slojeva se primenjuje nakon rezidualne konekcije:

$$\text{Output} = \text{LayerNorm}(\text{Layer}(x) + x).$$

Ove tehnike osiguravaju da model efikasno trenira, održava stabilne gradijente i omogućava korišćenje dubljih neuronskih mreža bez značajnih problema sa performansama.

2.3.2 Dekoder

Dekoder je odgovoran za generisanje izlazne sekvence, poput prevođenja rečenice s jednog jezika na drugi. Deli sličnosti sa enkoderom, ali uvodi dodatne mehanizme za obradu sekvencijalnih podataka i za obraćanje pažnje na izlaz enkodera.

Dekoder ima tri glavne komponente:

- Maskirani *Self-Attention* Mehanizam
- *Enkoder-Dekoder Attention* Mehanizam
- *Feed-Forward* Neuronska Mreža (**FFN**, ranije **MLP**)

Ove komponente su kombinovane sa rezidualnim konekcijama i normalizacijom slojeva, slično kao kod enkodera [29].

Maskirani Self-Attention Mehanizam

Self-attention mehanizam u dekoderu je sličan onom u enkoderu, ali sa ključnom razlikom: **maskiranjem** (*Slika 17*). Pri generisanju izlazne sekvence, svaka pozicija može da se odnosi samo na prethodne pozicije, ne i na buduće. Ovo sprečava dekoder da "vara" gledajući unapred u tokene koji još nisu generisani.

Attention težine se računaju kao:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V$$

gde:

- Q, K, i V su matrice upita, ključeva i vrednosti,
- d_k je dimenzija ključeva,
- M je matrica maske koja postavlja *attention* težine na $-\infty$ za nedozvoljene pozicije (one koje ne bi trebalo da budu obrađene).

	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
CAT	0.124	0.278	$-\infty$	$-\infty$	$-\infty$	$-\infty$
IS	0.147	0.132	0.262	$-\infty$	$-\infty$	$-\infty$
A	0.210	0.128	0.206	0.212	$-\infty$	$-\infty$
LOVELY	0.146	0.158	0.152	0.143	0.227	$-\infty$
CAT	0.195	0.114	0.203	0.103	0.157	0.229

Slika 17: Izgled $\frac{QK^T}{\sqrt{d_k}} + M$ matrice pre primene *softmax* funkcije [30]

Enkoder-Dekoder Attention

Nakon računanja maskiranog *self-attention* mehanizma, dekoder obraća pažnju na izlaz enkodera koristeći standardni *attention* mehanizam. Ovo omogućava dekoderu da se fokusira na relevantne delove ulazne sekvence dok generiše izlaz.

Attention mehanizam ovde se može opisati kao:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

U ovom slučaju, Q dolazi iz prethodnog sloja dekodera, dok K i V dolaze iz završnih slojeva enkodera.

Feed-Forward Neuronska Mreža (FFN)

Nakon *attention* slojeva, izlaz prolazi kroz **FFN** mrežu, identičnu onoj koja se koristi u enkoderu. Ovaj modul se sastoji od dve linearne transformacije sa $ReLU$ aktivacijom između njih [19]:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

gde:

- W_1, W_2 su težinske matrice, a b_1, b_2 su pomeraji,
- $ReLU$ aktivacija uvodi nelinearnost.

Poslednji Linearni i Softmax Sloj

Nakon prolaska kroz više dekoderskih modula, konačni izlaz je sekvenca vektora, jedan za svaku poziciju u ciljnoj sekvenci. Ovi vektori se transformišu linearno i zatim prolaze kroz *softmax* funkciju kako bi generisali verovatnoće ciljnog vokabulara:

$$P(\text{trenutna reč} \mid \text{prethodne reči}) = \text{softmax}(W_o \cdot h + b_o)$$

gde:

- W_o je težinska matrica izlaza,
- h je skriveno stanje iz poslednjeg dekoderskog sloja,
- b_o je pomeraj.

Dekoder generiše izlaznu sekvencu jedan po jedan token, vraćajući prethodno generisane tokene u model sve dok cela sekvenca nije izgenerisana [29].

2.4 Generisanje

Tokom generisanja, transformer generiše predikcije za sekvencu na osnovu zadatog ulaza. Ključni aspekt generisanja je da model obrađuje svaki token u sekvenci jedan po jedan, koristeći prethodno generisane tokene za generisanje sledećeg. U ovom scenariju, složenost je obično linearna u odnosu na dužinu sekvence n.

- **Vremenska složenost:** $O(n)$
- **Opis:** Složenost proizlazi iz obrade svakog tokena u sekvenci pojedinačno.

2.5 Trening

Transformer obrađuje celu sekvencu odjednom u toku treninga, koristeći paralelizam. Sve pozicije se obrađuju simultano, što znači da je složenost konstantna u odnosu na dužinu sekvence. Međutim, sam *attention* mehanizam uključuje izračunavanje međusobnih odnosa, što dovodi do kvadratne složenosti u odnosu na dužinu sekvence.

- **Vremenska složenost:** $O(1)$ za paralelnu obradu svakog tokena
- **Opis:** Trening koristi paralelnu obradu, ali ukupna složenost *attention* mehanizma je $O(n^2)$ zbog međusobnih poređenja kroz dužinu sekvence.

3 Vision Transformeri

3.1 Istorija i razvoj

Vision transformer (**ViT**) je predstavljen u radu *"An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale"* [7] autora Dosovitskiy i ostalih saradnika (2020). Primarna inovacija **ViT**-a je prilagođavanje arhitekture transformera, prvobitno dizajnirane za **NLP**, za rad sa vizuelnim podacima. Ova promena označava odstupanje od tradicionalnih **CNN**-ova, koristeći prednosti *self-attention* mehanizama za obradu slika.

3.2 Poređenje sa CNN-ovima

- **Induktivna pretpostavka:**
  - **CNN:** Ima jaku induktivnu pretpostavku zbog svojih konvolucionih filtara, koji nameću lokalne prostorne hijerarhije, kao i *pooling* slojeva koji omogućavaju translacionu invarijantnost. To znači da **CNN**-ovi pretpostavljaju da su lokalne karakteristike (npr. ivice) važne, pa pošto filteri dele težine, mreža traga za tim obrascima unutar čitave slike.
  - **ViT:** Nema ugrađenu ovu pretpostavku. Umesto toga, uči odnose između svih delova slike putem *self-attention* mehanizma, što mu omogućava da modeluju globalne zavisnosti i interakcije između različitih delova slike [7].
- **Receptivno Polje:**
  - **CNN:** Ima ograničeno receptivno polje, što znači da svaki filtar pokriva samo malu, lokalizovanu oblast slike. Da bi se modelovao globalni kontekst, potrebni su dublji slojevi ili veća receptivna polja, što povećava računsku složenost.
  - **ViT:** Poredi svaki deo slike u odnosu na sve druge delove zahvaljujući *self-attention* mehanizmu. Ovo znači da može modelovati udaljene zavisnosti od samog početka, bez potrebe za dubokim ili složenijim arhitekturama [7].

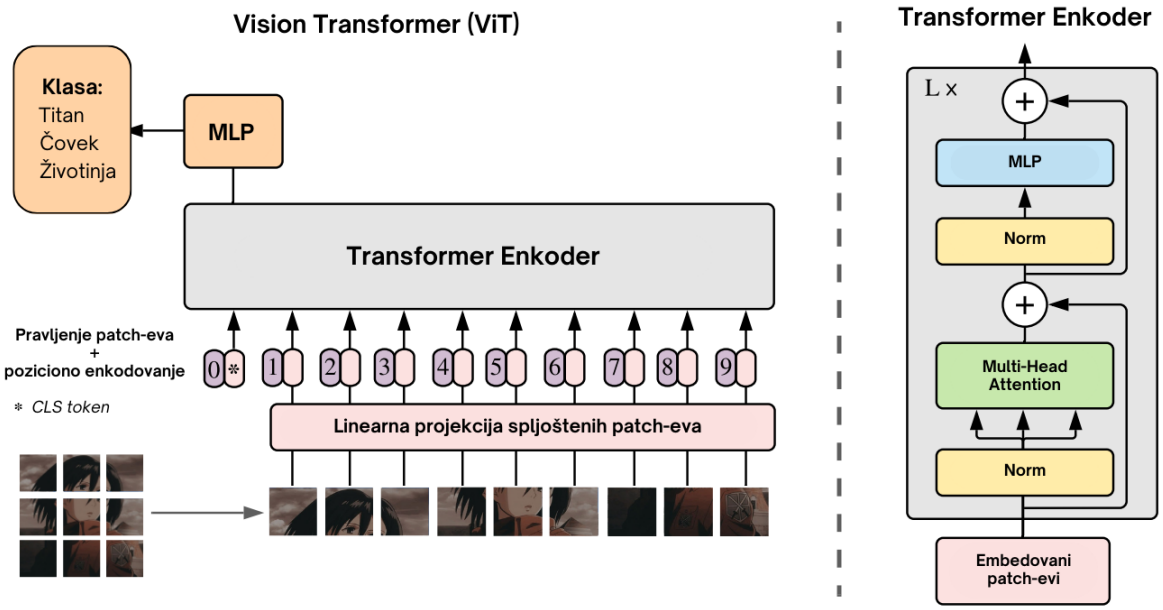
3.3 Prednosti i Nedostaci

- **Prednosti:**
  - **Globalni Kontekst:** Mogu da modeluju globalni kontekst i odnose između svih delova slike od samog početka, zahvaljujući *self-attention* mehanizmu.
  - **Fleksibilnost:** Nisu ograničeni na lokalnu obradu kao **CNN**-ovi i mogu da nauče reprezentacije koje nisu ograničene strukturom konvolucionih filtara.
  - **Skalabilnost:** Mogu se efikasno skalirati povećanjem podataka i računarskih resursa. Vrlo lako se dotreniraju za specifične zadatke sa malim skupom podataka koristeći prethodno obučene velike modele [7].
- **Nedostaci:**
  - **Efikasnost Podataka:** Obično zahtevaju velike količine podataka za efikasnu obuku. Nemaju istu induktivnu pretpostavku kao **CNN**-ovi, što može dovesti do lošijih performansi sa ograničenim podacima [7].
  - **Računski Trošak:** Kvadratna složenost *self-attention* mehanizma može biti računski skupa, posebno za slike visoke rezolucije ili duge nizove tokena.
  - **Složenost Obuke:** Obuka može biti izazovna i često zahteva pažljivo podešavanje hiperparametara i velike računarske resurse.

3.4 Arhitektura Vision Transformera

Glavni delovi **ViT** [7] arhitekture (*Slika 18*) su:

- **Tokenizacija slike**
- **Poziciono enkodovanje**
- **Transformer Enkoder**
- **Klasifikacioni MLP**[3]



Slika 18: Arhitektura Vision Transformera za klasifikaciju slika

[3]Rešavamo isti problem kao i kod **CNN**-ova, u opštem slučaju arhitektura nakon enkodera može biti proizvoljna.

3.4.1 Tokenizacija slike

Kod Vision Transformera, proces tokenizacije slike je ključan za prilagođavanje originalnoj arhitekturi transformera, kako bi ona radila sa slikama. Za razliku od konvolucionih neuronskih mreža (**CNN**), koje obrađuju slike u celini, **ViT** deli sliku na manje delove poznate kao *patch*-evi [7].

Deljenje slike na patch-eve

Prvi korak u tokenizaciji predstavlja deljenje ulazne slike na *patch*-eve fiksne veličine. Razmotrimo ulaznu sliku dimenzija $H \times W \times C$, gde su H i W visina i širina slike, a C je broj kanala boje (obično 3 za RGB slike). Slika se deli na nepreklapajuće *patch*-eve, svaki veličine $P \times P \times C$.

Ukupan broj *patch*-eva N dat je formulom:

$$N = \frac{HW}{P^2}.$$

Svaki *patch* se zatim "spljošti" u jednodimenzionalni vektor veličine P^2C [33].

Embedobanje patch-eva

Nakon što je slika podeljena na *patch*-eve i svaki *patch* je spljošten, sledeći korak je da se ovi vektori projektuju u višedimenzionalni prostor koji transformerov enkoder može da obradi. Ovo se ostvaruje putem **linearne transformacije**, ovakav sloj se često naziva embeding sloj (eng. *patch embedding layer*).

Za dati spljošteni *patch* $x_i \in \mathbb{R}^{P^2C}$, embeding sloj ga projektuje u vektor dimenzije d_{model}, gde je d_{model} dimenzionalnost ulaznog prostora enkodera:

$$z_i = x_i \cdot W_e + b_e.$$

Ovde je $W_e \in \mathbb{R}^{(P^2C)\times d_{model}}$ matrica težina koja se uči, a $b_e \in \mathbb{R}^{d_{model}}$ je pomeraj. Ova transformacija konvertuje svaki *patch* u token, vektor dimenzije d_{model}, pogodan za ulaz u transformer enkoder.

Rezultujuća sekvenca embedovanih *patch*-eva formira ulaz u transformer enkoder, omogućavajući modelu da obradi sliku na sličan način kao što transformer obrađuje sekvencu reči. Redosled i prostorni odnosi između *patch*-eva se čuvaju dodavanjem pozicionih vektora, koji se sabiraju sa ovim vektorima pre nego što prođu kroz enkoder [34].

3.4.2 Poziciono Enkodovanje

U **ViT**-ovima, kao i u klasičnim transformerima, redosled ulaznih tokena je ključan za očuvanje prostorne strukture podataka. Međutim, tokeni u **ViT** modelu sami po sebi nemaju nikakvu inherentnu informaciju o poziciji, jer su rezultati deljenja slike na *patch*-eve. Da bi model mogao da iskoristi prostornu informaciju o *patch*-evima, neophodno je dodati informacije o poziciji svakom tokenu [7].

Fiksni pozicioni vektori

U **ViT** modelu, pozicioni vektori se često generišu korišćenjem sinusnih i kosinusnih funkcija različitih frekvencija (isto kao i kod originalnog transformera). Za svaki *patch*, pozicioni vektor p_i dimenzije d_{model} se dodaje vektoru *patch*-a z_i kako bi se formirao finalni ulaz u Transformer enkoder:

$$z_i' = z_i + p_i.$$

Poziciono enkodovanje je definisano sledećim formulama:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

gde je *pos* pozicija *patch*-a u sekvenci, a i indeks dimenzije. Iako bi imalo smisla koristiti 2D poziciono enkodovanje za slike, ovaj pristup ne daje bolje rezultate u praksi [7], pa se koristi jednodimenzionalni pristup.

Pozicioni vektori koji se uče

Umesto korišćenja fiksnih sinusnih i kosinusnih funkcija, moguće je i da se pozicioni vektori nauče tokom treniranja modela. U ovom pristupu, pozicioni vektori se tretiraju kao dodatni parametri koji se optimizuju zajedno sa težinama modela. Međutim, istraživanja su pokazala da fiksni pozicioni vektori daju rezultate slične naučenim vektorima, dok su jednostavniji za implementaciju i računanje [7].

3.4.3 Transformer Enkoder

ViT koristi arhitekturu Transformer enkodera, koja je već detaljno objašnjena. U kontekstu **ViT**-a, enkoder obrađuje sekvencu tokena slike kako bi uhvatio odnose i zavisnosti između različitih *patch*-eva slike.

Transformer enkoder može imati više slojeva, a sastavni deo svakog sloja su:

- **Multi-Head Attention:** omogućava modelu da obrati pažnju na različite delove slike istovremeno, modelujući složene odnose između *patch*-eva,
- **Feedforward Neuronska Mreža:** Primenjuje se nezavisno na svaki token i dalje obrađuje informacije dobijene od *self-attention* mehanizma,
- **Rezidualne Konekcije i Normalizacija Sloja:** Ovi elementi pomažu u stabilizaciji obuke i poboljšanju konvergencije normalizovanjem aktivacija i uključivanjem rezidualnih konekcija.

Izlaz iz enkodera, posebno reprezentacija `[CLS]` tokena, koristi se za zadatke klasifikacije slika [7].

3.4.4 Klasifikacioni MLP

Finalni output se dobija kroz *Multi-Layer Perceptron* (MLP) glavu, koja obrađuje reprezentaciju `[CLS]` token-a.

`[CLS]` token je poseban token koji se dodaje sekvenci tokena slika pre nego što se prosledi enkoderu. Njegova svrha je da agregira informacije iz svih *patch*-eva i služi kao kompresovana reprezentacija cele slike. Konačna reprezentacija ovog tokena, nakon prolaska kroz enkoder, koristi se za klasifikaciju.

Upotreba `[CLS]` tokena u **ViT**-u je inspirisana modelom **BERT** [22] u obradi prirodnog jezika, gde služi sličnu svrhu u agregaciji kontekstualnih informacija za zadatke klasifikacije. Iako je `[CLS]` token pokazao svoju efikasnost u **ViT**-ovima, treba napomenuti da arhitektura može funkcionisati i bez njega. Alternativne metode, kao što su *pooling* metode ili neke druge strategije, mogle bi postići slične rezultate, ali je `[CLS]` token ostao standardan pristup zbog lakše i efikasnije implementacije [7].

4 Eksperimenti i Rezultati

4.1 Postavka eksperimenta

4.1.1 Softver

Za potrebe eksperimenata korišćeni su:

- **Framework:** PyTorch
- **Jezik:** Python
- **Biblioteke:** numpy, torchvision, matplotlib, seaborn

Upotrebljen je model sa dimenzijom tokena od 32 i korišćena su 3 enkoderska bloka. **MLP** unutar enkodera imao je dimenziju od 32. Brzina učenja je postavljena na 0.005, a za obuku je korišćen **Adam** optimizator [35]. Veličina *batch*-a bila je 128, a model je treniran tokom 30 epoha. Kao funkcija gubitka korišćen je *Cross entropy loss*.

4.1.2 Hardver

Treniranje modela je obavljeno na laptopu sa sledećom hardverskom konfiguracijom:

- **GPU:** NVIDIA GeForce GTX 1650 Ti sa 4 GB RAM-a.
- **CPU:** Intel i7-10750H (12 jezgara) @ 5.000GHz.
- **RAM:** 64 GB.

Zbog resursa i vremenskih ograničenja, kao i zbog resursno intenzivne prirode treniranja modela na velikim skupovima podataka, direktna poređenja sa najmodernijim modelima na velikim skupovima podataka nisu bila izvodljiva. Umesto toga, ovi eksperimenti se fokusiraju na implementacione detalje **ViT**-a, različite delove njegove arhitekture i performanse kroz kontrolisane eksperimente na manjim skupovima podataka.

Jedan od implementacionih detalja koji je omogućio efikasno izvršavanje[4] jeste korišćenje PyTorch-ovih funkcija za deljenje slika na *patch*-eve i međusobno paralelno množenje tokena unutar *multi-head attention* bloka. Ove izmene su omogućile značajno ubrzanje (12x do 15x) u odnosu na referentnu implementaciju [33].

4.1.3 Dataset

Eksperimenti su sprovedeni nad MNIST [36] skupom podataka, koji se sastoji od 60.000 crno-belih slika rukom pisanih cifara za obuku i 10.000 slika za testiranje. Svaka slika je veličine 28x28 piksela. Ovaj dataset je je korišćen zbog svoje veličine i mogućnosti da se **ViT** testira na zadacima manjih razmera.

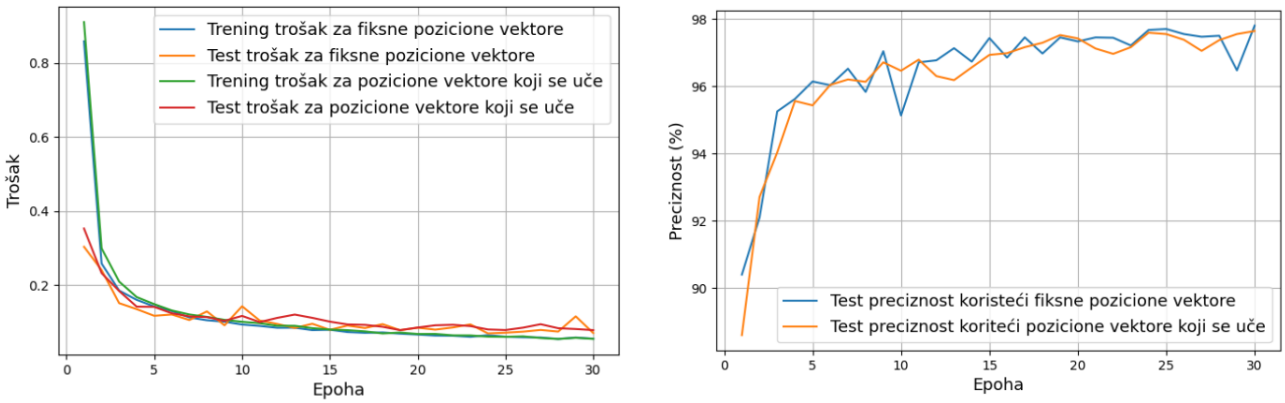
[4] Kod je dostupan na `https://github.com/Kovelja009/ViT_implementation`

4.2 Eksperiment 1: Poređenje fiksnih i pozicionih vektora koji se uče

Cilj ovog eksperimenta jeste poređenje performansi fiksnih i pozicionih vektora koji se uče u pogledu preciznosti klasifikacije i njihove sposobnosti da modeluju prostorne odnose na slikama.

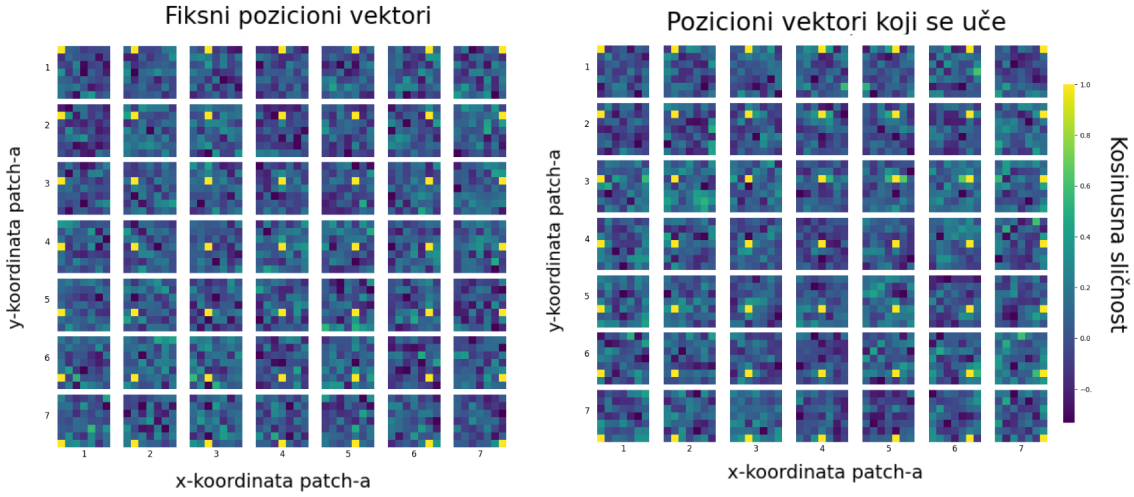
Rezultati:

- **Poređenje Performansi:** Rezultati sa *Slike 19* pokazuju da oba tipa vektora daju sličnu preciznost klasifikacije. Međutim, pozicioni vektori koji se uče pružaju nešto stabilniju preciznost kroz trening sesiju.



Slika 19: Trošak tokom treninga i testa (levo) i preciznost tokom testa (desno) za fiksne i pozicione vektore koji se uče

- **Prostorni Odnosi:** *Slika 20* prikazuje kosinusnu sličnost između različitih delova slike. Ova analiza otkriva da pozicioni vektori koji se uče modeluju prostorne odnose između delova slike tako, da oni na kraju podsećaju na fiksne pozicione vektore, čime je zadatak uspešno naučen.



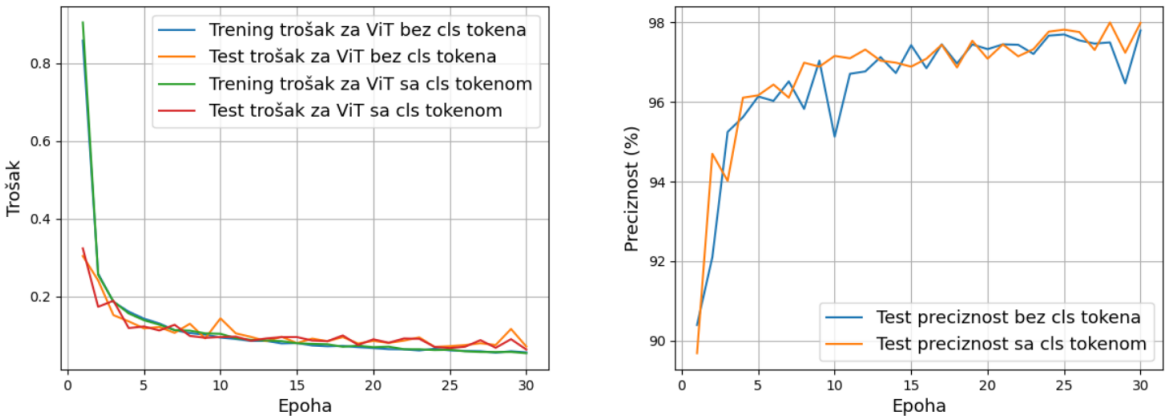
Slika 20: Kosinusna sličnost između različitih *patch*-eva za fiksne i pozicione vektore koji se uče

4.3 Eksperiment 2: Uticaj uklanjanja CLS tokena

Cilj ovog eksperimenta je ispitivanje uticaja uklanjanja `[CLS]` tokena na performanse **ViT**-a.

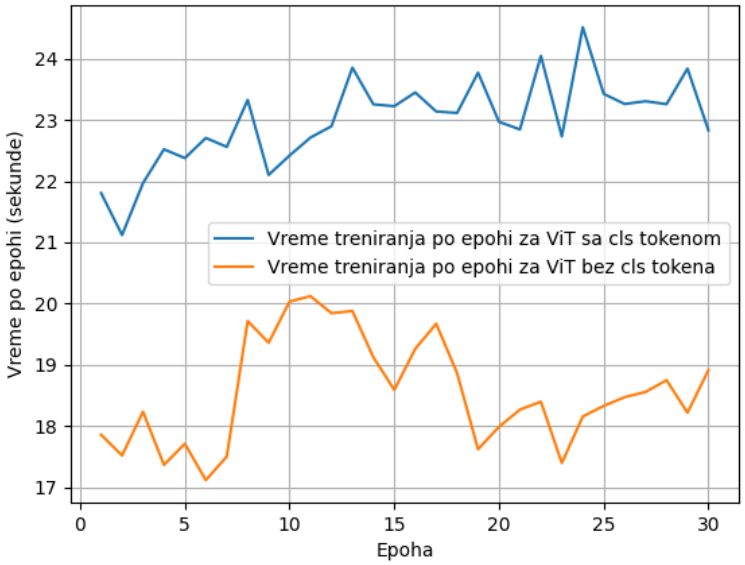
Rezultati:

- **Performanse Modela:** *Slika 21* potvrđuje da **ViT** ima skoro identične performanse sa i bez `CLS` tokena, što je u skladu sa hipotezom koja je predstavljena u originalnom radu [7].



Slika 21: Trošak tokom treninga i testa (levo) i preciznost tokom testa (desno) za modele sa i bez `CLS` tokena

- **Vreme Treninga:** *Slika 22* otkriva značajno smanjenje vremena treninga po epohi kada se `CLS` token izostavi (približno 1.3 puta brži). Pretpostavka je da je to zbog načina na koji je `CLS` token implementiran i ostaje predmet dalje istrage.

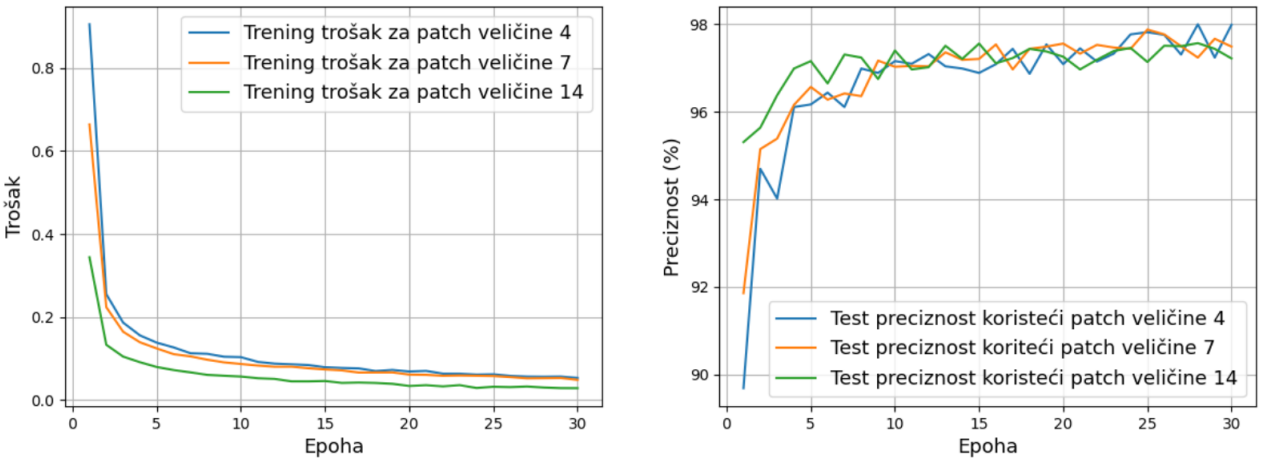


Slika 22: Trošak tokom treninga i testa (levo) i preciznost tokom testa (desno) za modele sa i bez `CLS` tokena

4.4 Eksperiment 3: Uticaj veličine patch-eva na performanse

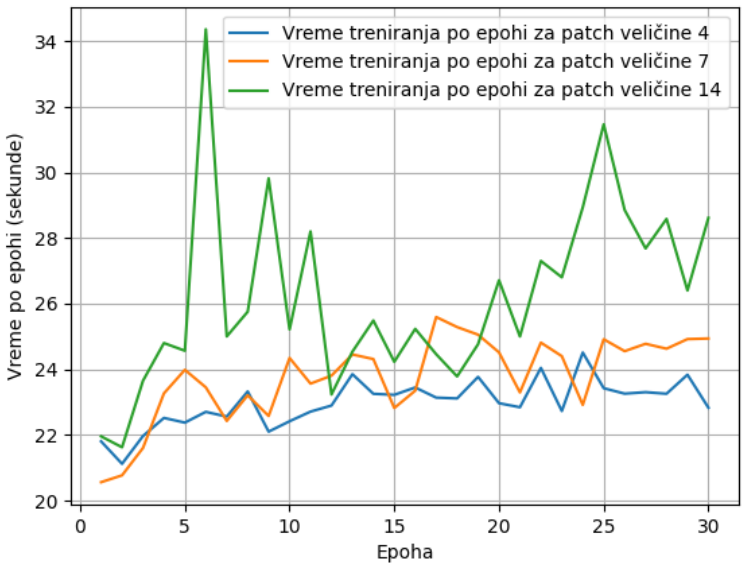
Cilj trećeg eksperimenta je da se proceni uticaj različitih veličina *patch*-eva na performanse modela. Testirane su tri veličine patch-eva: 7x7, 4x4 i 2x2.

- **Performanse:** Kao što je prikazano na *Slici 23*, sve tri veličine *patch*-eva su dale slične gubitke i preciznost tokom obuke, što ukazuje na to da veličina *patch*-a ne utiče značajno na sposobnost modela da uči i postiže dobre performanse na **MNIST** skupu podataka. Zapažanje je da je model sa veličinom *patch*-a 14x14 najmanje naučio u poređenju sa početnim epohama. Intuicija je da se model preprilagođava pošto je *patch* veliki (2x2 ukupan broj *patch*-eva), pa je bolje koristiti manje *patch*-eve.



Slika 23: Trošak tokom treninga (levo) i preciznost tokom testa (desno) za modele sa različitom veličinom patch-eva

- **Vreme obuke:** *Slika 24* otkriva zanimljivo zapažanje: *patch*-evi veličine 14 (2x2) imali su najsporije vreme obuke po epohi. Razlog za to može biti posledica implementacije linearnog *embedding* sloja, mada su potrebna dodatna ispitivanja da bi se ova hipoteza potvrdila.



Slika 24: Vreme obuke po epohi za različite veličine *patch*-eva

5 Zaključak

U radu je detaljno objašnjena arhitektura Vision Transformer modela, uključujući proces pretvaranja slika u tokene, korišćenje enkodera, i značaj klasičnog transformer modela.

Kroz implementaciju i eksperimentalne rezultate, pokazano je da **ViT** modeli mogu uspešno da obrade slike i postignu konkurentne rezultate.

Eksperimenti su obuhvatili upotrebu fiksnih nasuprot pozicionim vektorima koji se uče, uticaj uklanjanja `[CLS]` tokena na performanse modela, i analizu uticaja veličine *patch*-eva na vreme obuke i preciznost modela.

Rezultati su pokazali da fiksni i pozicioni vektori koji se uče imaju slične performanse, dok je uklanjanje `[CLS]` tokena imalo minimalan uticaj na performanse, ali je značajno smanjilo vreme obuke. Takođe, veličina *patch*-eva ima uticaj na vreme obuke, s manjim *patch*-evima koji mogu da unaprede efikasnost modela.

Potencijalni fokus budućih istraživanja bi bio na isprobavanju alternativnih strategija za modelovanje globalnog konteksta bez oslanjanja na `[CLS]` token. Ovo može uključivati eksperimentisanje sa različitim *pooling* strategijama. Takođe, vizualizacija *attention* mapa tokom procesa klasifikacije mogla bi pružiti uvide u delove slike na koje se enkoder fokusira, što bi dovelo do bolje interpretabilnosti modela.

Literatura

[1] Li Fei-Fei. *With spatial intelligence, AI will understand the real world.* 2024. URL: https://www.ted.com/talks/fei_fei_li_with_spatial_intelligence_ai_will_understand_the_real_world.

[2] Yan LeCun. "Backpropagation applied to handwritten zip code recognition". In: *Neural Computation* 1 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". In: NIPS'12 (2012), pp. 1097–1105.

[4] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: (2015).

[5] Kaiming He et al. "Deep residual learning for image recognition". In: (2016), pp. 770–778.

[6] Embedl. *Vision Transformers vs CNNs at the Edge.* 2024. URL: https://www.embedl.com/knowledge/vision-transformers-vs-cnns-at-the-edge.

[7] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: (2021). arXiv: 2010.11929 [cs.CV]. URL: https://arxiv.org/abs/2010.11929.

[8] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[9] Daksh Bhatnagar. *History of CNN and its impact in the field of Artificial Intelligence.* 2023. URL: https://medium.com/accredian/history-of-cnn-its-impact-in-the-field-of-artificial-intelligence-2b1efb7d99e5.

[10] Avishek Biswas. *The History of Convolutional Neural Networks for Image Classification (1989 - Today).* 2024. URL: https://towardsdatascience.com/the-history-of-convolutional-neural-networks-for-image-classification-1989-today-5ea8a5c5fe20.

[11] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: (2009), pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[12] Giorgio Roffo. "Ranking to Learn and Learning to Rank: On the Role of Ranking in Pattern Recognition Applications". In: (June 2017).

[13] Zoumana Keita. *An Introduction to Convolutional Neural Networks (CNNs)*. 2023. URL: https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns.

[14] Yulia Gavrilova. *Convolutional Neural Networks for Beginners*. 2021. URL: https://serokell.io/blog/introduction-to-convolutional-neural-networks.

[15] savyakhosla. *CNN | Introduction to Padding*. 2023. URL: https://www.geeksforgeeks.org/cnn-introduction-to-padding/.

[16] deepai. *Stride (Machine Learning)*. 2024. URL: https://deepai.org/machine-learning-glossary-and-terms/stride.

[17] Muhamad Yani, S Irawan, and Casi Setianingsih. "Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail". In: *Journal of Physics: Conference Series* 1201 (May 2019), p. 012052. DOI: 10.1088/1742-6596/1201/1/012052.

[18] Sejai Jaiswai. *Multilayer Perceptrons in Machine Learning: A Comprehensive Guide*. 2024. URL: https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning.

[19] Ashish Vaswani et al. "Attention Is All You Need". In: (2023). arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[20] David E. Rumelhart and James L. McClelland. *Learning Internal Representations by Error Propagation*. 1987, pp. 318–362.

[21] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[22] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: (2019). arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805.

[23] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: (2019).

[24] Sunke. *Brief history and motivation for the development of Transformer model*. 2024. URL: https://medium.com/@sunke1920/brief-history-and-motivation-for-the-development-of-transformer-model-920178ae4193.

[25] Indrajitbarat. *Recurrent Neural Networks (RNNs): Challenges and Limitations*. 2023. URL: https://medium.com/@indrajitbarat9/recurrent-neural-networks-rnns-challenges-and-limitations-4534b25a394c.

[26] Yao Ming et al. "Understanding Hidden Memories of Recurrent Neural Networks". In: (Oct. 2017).

[27] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[28] Ottavio Calzone. *An Intuitive Explanation of LSTM*. 2022. URL: https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c.

[29] Umar Jamil. *Attention is all you need (Transformer) - Model explanation (including math), Inference and Training*. 2023. URL: https://youtu.be/bCz4OMemCcA?si=g056Geiv1Z-UkKo7.

[30] Umar Jamil. *Attention is all you need (Transformer) - Model explanation (including math), Inference and Training*. 2023. URL: https://github.com/hkproj/transformer-from-scratch-notes.

[31] Abby Morgan. *Explainable AI: Visualizing Attention in Transformers*. 2023. URL: https://www.comet.com/site/blog/explainable-ai-for-transformers/.

[32] Florian June. *BatchNorm and LayerNorm*. 2023. URL: https://medium.com/@florian_algo/batchnorm-and-layernorm-2637f46a998b#:~:text=Difference%20between%20Batch%20Normalization%20and,all%20features%20within%20each%20sample..

[33] Brian Pulfer. *Vision Transformers from Scratch (PyTorch): A step-by-step guide*. 2022. URL: https://medium.com/@brianpulfer/vision-transformers-from-scratch-pytorch-a-step-by-step-guide-96c3313c2e0c.

[34] Aleksa Gordic. *Vision Transformer (ViT) - An image is worth 16x16 words | Paper Explained*. 2020. URL: https://youtu.be/j6kuz_NqkG0?si=oE9UTIuAGm2SLdm6.

[35] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: (2017). arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[36] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.