



**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

<b>1</b>	<b>Uvod</b>	<b>1</b>
1.1	Istorija i motivacija . . . . .	1
1.1.1	Rani Razvoj u Računarskom Vidu . . . . .	1
1.1.2	Uspon Dubokog Učenja . . . . .	2
1.1.3	Ograničenja CNN-ova . . . . .	2
1.1.4	Motivacija za uvođenje Vision Transformera . . . . .	2
1.2	Konvolucione Neuronske Mreže (CNN) . . . . .	3
1.2.1	Istorija CNN-ova . . . . .	3
1.2.2	Ljudski vizuelni sistem kao inspiracija . . . . .	4
1.2.3	Arhitektura CNN-ova . . . . .	6
<b>2</b>	<b>Transformeri</b>	<b>14</b>
2.1	Istorija i razvoj . . . . .	14
2.2	Motivacija za transformer arhitekturu . . . . .	14
2.2.1	Rekurentne Neuronske Mreže (RNN) . . . . .	15
2.2.2	Long Short-Term Memory Mreže (LSTM) . . . . .	16
2.2.3	Prednost transformera . . . . .	17
2.3	Arhitektura transformera . . . . .	17
2.3.1	Enkoder . . . . .	18
2.3.2	Dekoder . . . . .	26
2.4	Generisanje . . . . .	29
2.5	Trening . . . . .	29
<b>3</b>	<b>Vision Transformeri</b>	<b>30</b>
3.1	Istorija i razvoj . . . . .	30
3.2	Poređenje sa CNN-ovima . . . . .	30
3.3	Prednosti i Nedostaci . . . . .	31
3.4	Arhitektura Vision Transformera . . . . .	32
3.4.1	Tokenizacija slike . . . . .	33
3.4.2	Poziciono Enkodovanje . . . . .	34
3.4.3	Transformer Enkoder . . . . .	35
3.4.4	Klasifikacioni MLP . . . . .	36
<b>4</b>	<b>Eksperimenti i Rezultati</b>	<b>36</b>
4.1	Postavka eksperimenta . . . . .	36
4.1.1	Softver . . . . .	36

4.1.2	Hardver . . . . .	37
4.1.3	Dataset . . . . .	37
4.2	Eksperiment 1: Poređenje fiksnih i pozicionih vektora koji se uče . . . . .	38
4.3	Eksperiment 2: Uticaj uklanjanja CLS tokena . . . . .	40

# 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.

Na kraju, rad naglašava značaj Vision Transformera u oblasti računarskog vida, prikazujući njihov potencijal i ograničenja, kao i njihove praktične primene.

# 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 proдре 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. Radenje vida se smatra da je 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, a 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 (engl. *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 (CNNs)**, koje su predstavili Yann LeCun i drugi [2] krajem 1980-ih i početkom 1990-ih, revolucionisale su ovu oblast uvođenjem automatskog ekstraktovanja karakteristika kroz slojeve koji se uče (engl. *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 propaćeno 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 koja su motivisala potragu za novim pristupima. Jedan od značajnih nedostataka je njihova poteškoća u povezivanju 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 Transformer

Pojava **Vision Transformer (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. 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 (engl. *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.

## 1.2 Konvolucione Neuronske Mreže (CNN)

### 1.2.1 Istorija CNN-ova

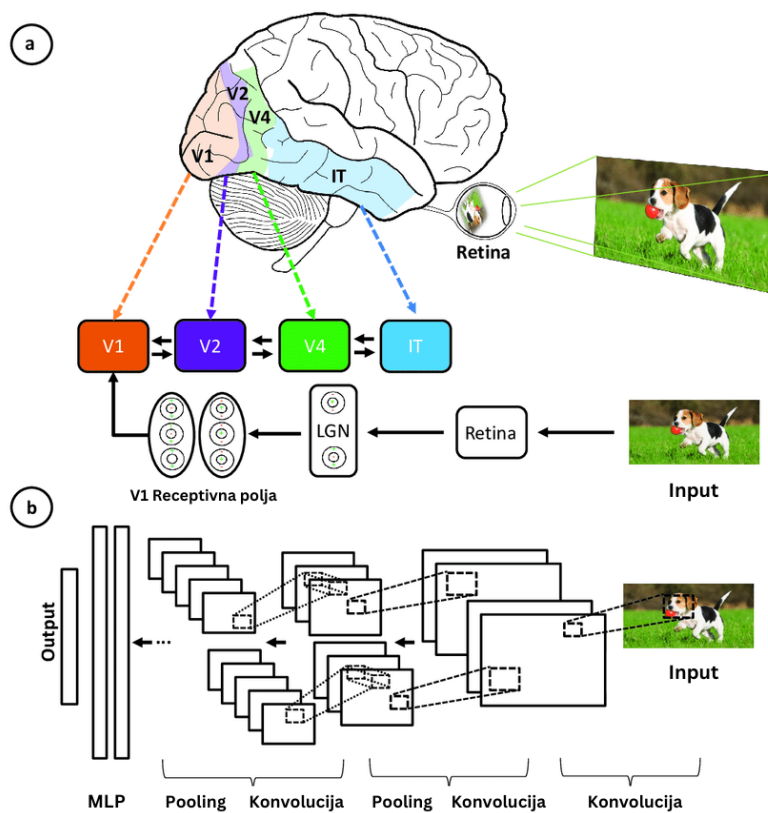
Konvolucione neuronske mreže su razvijene i prvi put korišćene oko 1980-ih. Tokom tog perioda, primarna primena **CNN**-ova bila je prepoznavanje rukom pisanih cifara, što je našlo praktičnu primenu u poštanskom sektoru za čitanje poštanskih i PIN kodova. Rani modeli **CNN**-ova, kao što je **LeNet** [6] koji je razvio Yann LeCun, pokazali su potencijal **CNN**-ova za zadatke prepoznavanja cifara.

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 ograničila upotrebu **CNN**-ova uglavnom na poštanski sektor, i tehnologija nije uspela da stekne širu primenu u oblasti mašinskog učenja.

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 nekoliko ključnih faktora: dostupnost velikih skupova podataka, kao što je **ImageNet** [7] skup sa milionima označenih slika, i značajna unapređenja u računarskim resursima, posebno **GPU**-ovima (engl. *graphics processing unit*). Ovi razvojni događaji omogućili su istraživačima da prevaziđu prethodna ograničenja i u potpunosti iskoriste mogućnosti konvolucionih neuronskih mreža.

### 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**. Pojedinačni neuroni reaguju na stimulse samo u ograničenom delu vizuelnog polja poznatom kao **receptivno polje** (engl. *receptive field*). Ova polja se preklapaju kako bi se pokrilo čitavo vizuelno područje.



Slika 1: Paralela između organizacije vizuelnog korteksa i CNN arhitekture

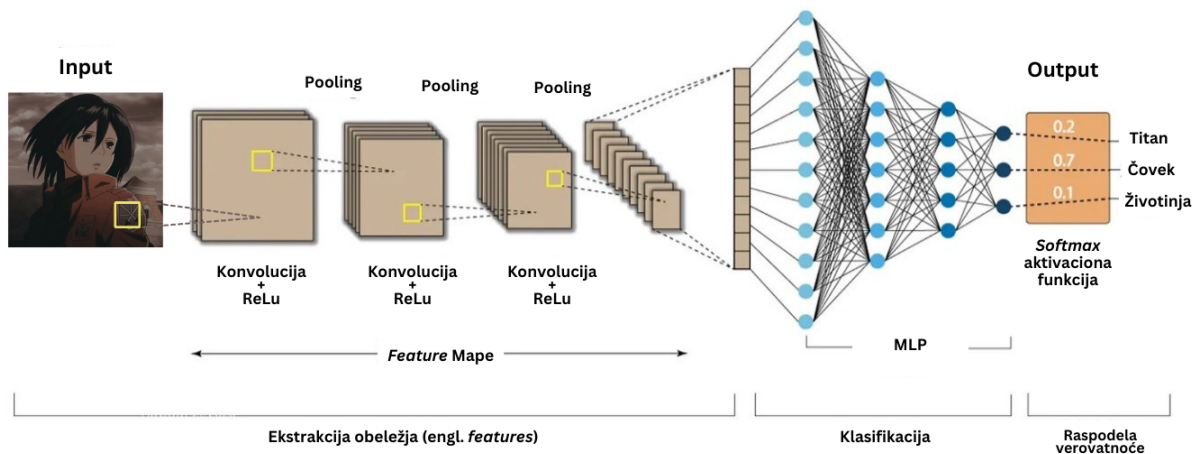


Glavne sličnosti [8] između organizacije vizuelnog korteksa i arhitekture CNN-ova su:

- **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:** Neuronu 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:** Neuronu 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 (engl. *kernels*) koji detektuju različite karakteristike u svakom konvolucionom sloju.
- **Nelinearnost:** Neuronu 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<sup>1</sup>, 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:

- **Konvolucionni slojevi**
- *Pooling* slojevi
- Višeslojni perceptron (engl. *Multi-Layer Perceptron* - MLP)

<sup>1</sup>Gradivni elementi CNN-a su identični, bez obzira na to da li je u pitanju problem regresije ili druge prirode; jedina razlika leži u MLP sloju

# 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 (pogledati sliku 3). Neka je  $K$  2D kernel sa dimenzijama  $f \times f$ , gde je  $f$  veličina kernela.

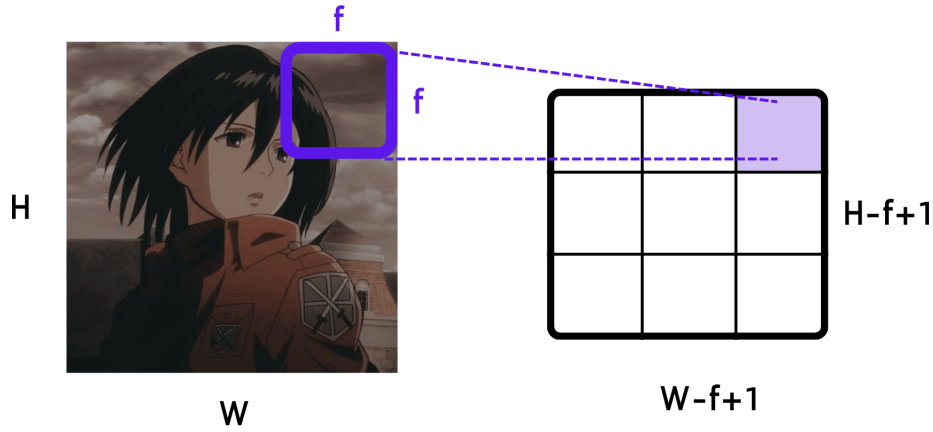
Konvolucija operacija se može matematički izraziti kao:

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

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.

## 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*. Veličina *feature* mape je smanjena u odnosu na ulaznu sliku zbog klizne operacije kernela.



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

### 3. Popunjavanje (engl. Padding)

Popunjavanje se koristi za kontrolu prostornih dimenzija izlazne *feature* mape. Popunjavanje dodaje dodatne piksele oko ivica ulazne slike. Neka je  $p$  veličina popunjavanja. Popunjena ulazna slika  $I'$  ima dimenzije  $(H + 2p) \times (W + 2p)$ .

Konvolucija operacija 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 (engl. Stride)

Korak određuje koliko piksela se filter pomera pri svakom koraku tokom konvolucije. Neka je  $s$  vrednost koraka. Korak utiče na dimenzije izlazne karakteristične mape. Sa korakom  $s$ , izlazna karakteristična mapa  $F$  ima dimenzije:

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

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

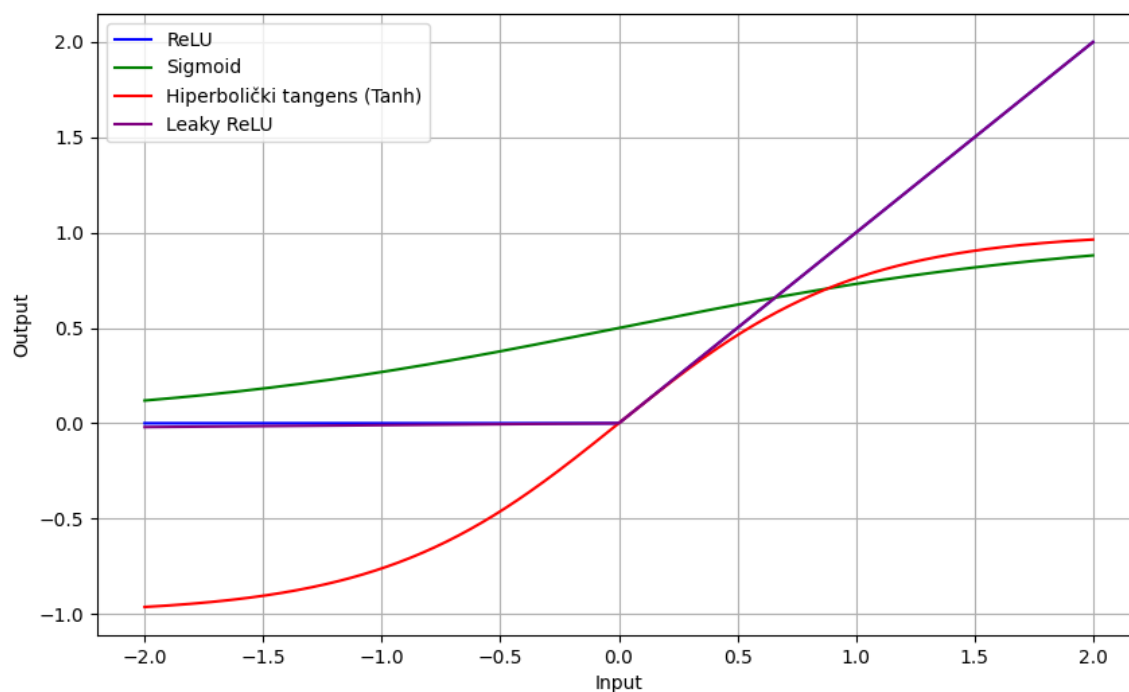
gde su  $H_{\text{out}}$  i  $W_{\text{out}}$  visina i širina izlazne *feature* mape, respektivno.

## 5. Aktivaciona Funkcija

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

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

gde je  $x$  ulaz u aktivacionu funkciju. Neke od najčešćih 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 (engl. *overfitting*). Najčešće korišćene operacije pooling-a su ***max pooling*** i ***average pooling***.

### 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*-a, 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  $s$ , operacija *max pooling*-a se može izraziti kao:

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

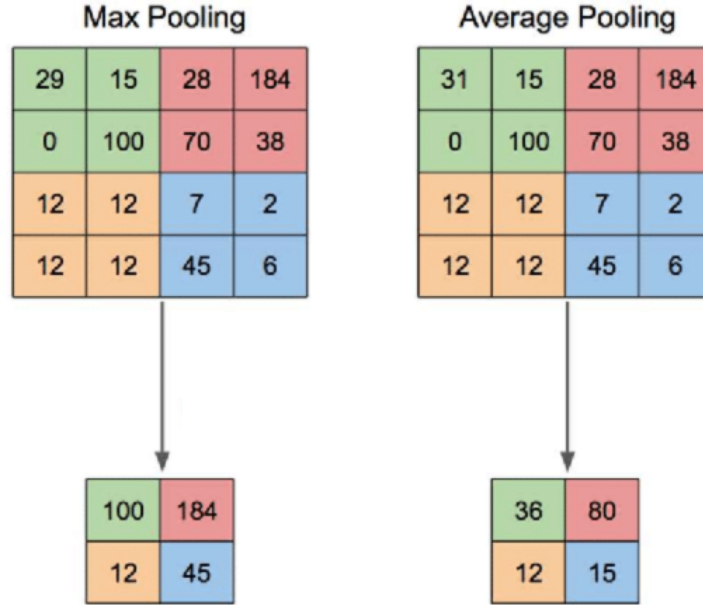
gde  $M(i, j)$  predstavlja vrednost izlazne *feature* mape na poziciji  $(i, j)$ .

### Average Pooling

Kod *average pooling*-a, 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  $s$ , operacija *average pooling*-a 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  $(i, j)$ .



Slika 5: Primer izvođenja *max pooling* i *average pooling* operacije

## Izlaz Pooling-a

Dimenzije izlazne *feature* mape zavise od veličine prozora za pooling  $f$  i koraka  $s$ . Data ulazna *feature* mapa  $F$  sa dimenzijama  $H \times W$ , 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  $H_{\text{out}}$  i  $W_{\text{out}}$  visina i širina izlazne *feature* mape, respektivno.

## 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*).

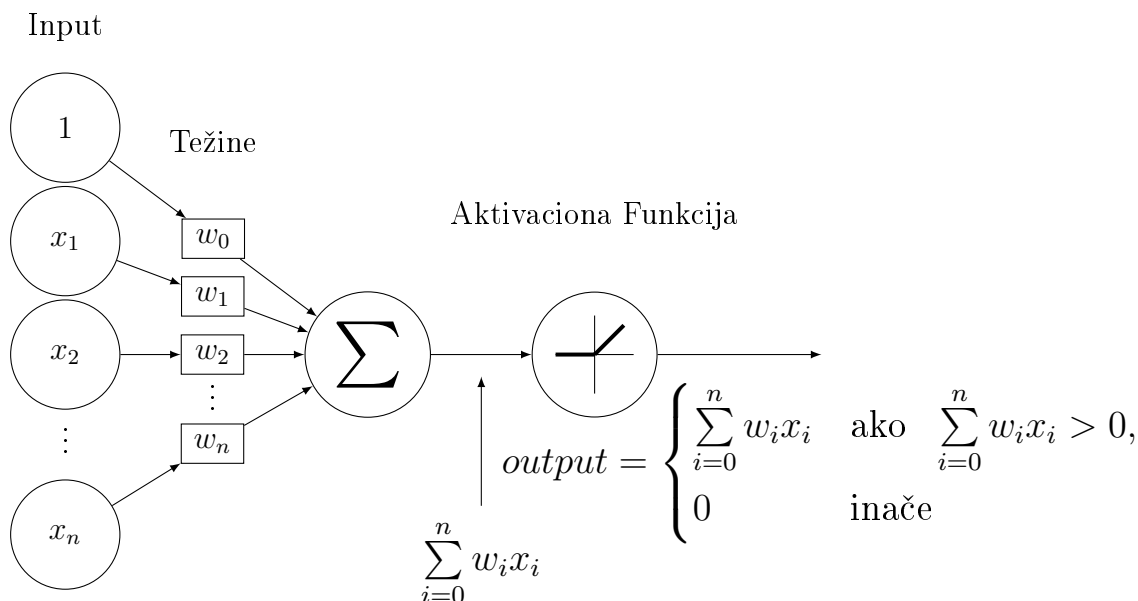
Za dat ulazni vektor  $\mathbf{x} = [x_1, x_2, \dots, x_n]$  i odgovarajuće težine  $\mathbf{w} = [w_1, w_2, \dots, w_n]$ , perceptron računa ponderisanu sumu na sledeći način:

$$z = \sum_{i=1}^n w_i x_i + b$$

gde je  $b$  (*bias*) slobodan član.

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.

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

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

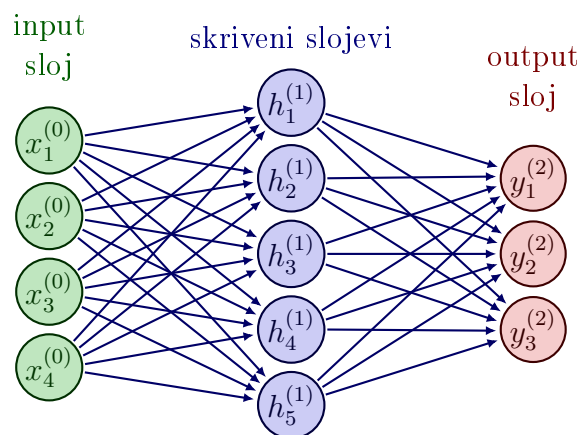
gde:

- $\mathbf{h}^{(0)} = \mathbf{x}$  je ulazni vektor,
- $\mathbf{W}^{(l)}$  i  $\mathbf{b}^{(l)}$  su matrica težina i vektor pristrasnosti 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 verovatnoće 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*" [9] od Vaswani-ja i saradnika 2017. godine, označio je značajan napredak u oblasti obrade prirodnog jezika (**NLP**). Pre toga, modeli kao što su *Recurrent Neural Networks* (**RNNs**) [10] i *Long Short-Term Memory Networks* (**LSTMs**) [11] 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*) [12] i **GPT** (*Generative Pre-trained Transformer*) [13], 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.

### 2.2.1 Rekurentne Neuronske Mreže (RNN)

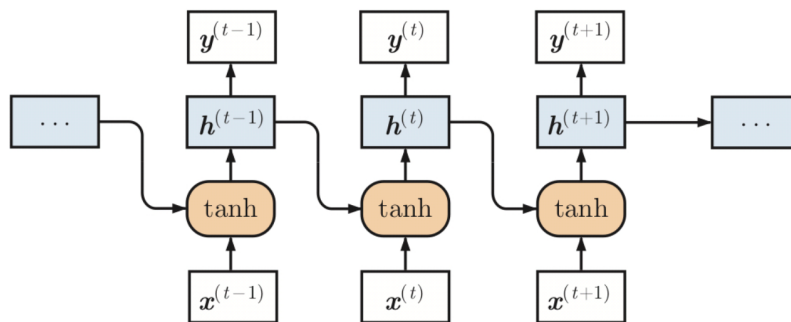
RNN-ovi procesiraju sekvence podataka održavanjem skrivenog stanja (engl. *hidden state*) koje čuva informacije o prethodnim elementima u sekvenci (slika 8). Skriveno stanje u vremenskom koraku  $t$ , označeno kao  $h_t$ , se računa kao:

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

gde:

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

RNN-ovi pate od problema nestajućih i eksplodirajućih gradijenata (engl. *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.



Slika 8: Prikaz rekurentne neuronske mreže

### 2.2.2 Long Short-Term Memory Mreže (LSTM)

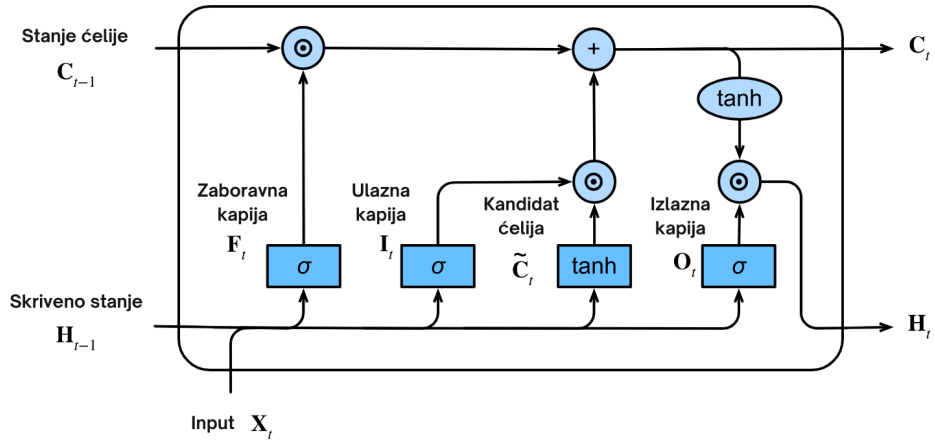
**LSTM**-ovi su uvedene da bi rešile neka od ograničenja **RNN**-ova. One uključuju memorijske ćelije i mehanizme za kontrolu protoka informacija, omogućavajući im da efikasnije hvataju udaljene zavisnosti. Ključne komponente **LSTM** ćelije uključuju ulaznu kapiju ( $i_t$ ), zaboravnu kapiju ( $f_t$ ), izlaznu kapiju ( $o_t$ ) i stanje ćelije ( $c_t$ ). Stanje ćelije se ažurira kao:

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

gde je  $\tilde{c}_t$  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 njihovog procesiranja znači da ne mogu efikasno koristiti paralelno računanje, što ograničava njihovu skalabilnost.



Slika 9: Prikaz arhitekture **LSTM** modela

### 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, hvatajući 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 modela da hvata različite aspekte odnosa između elemenata u sekvenci.

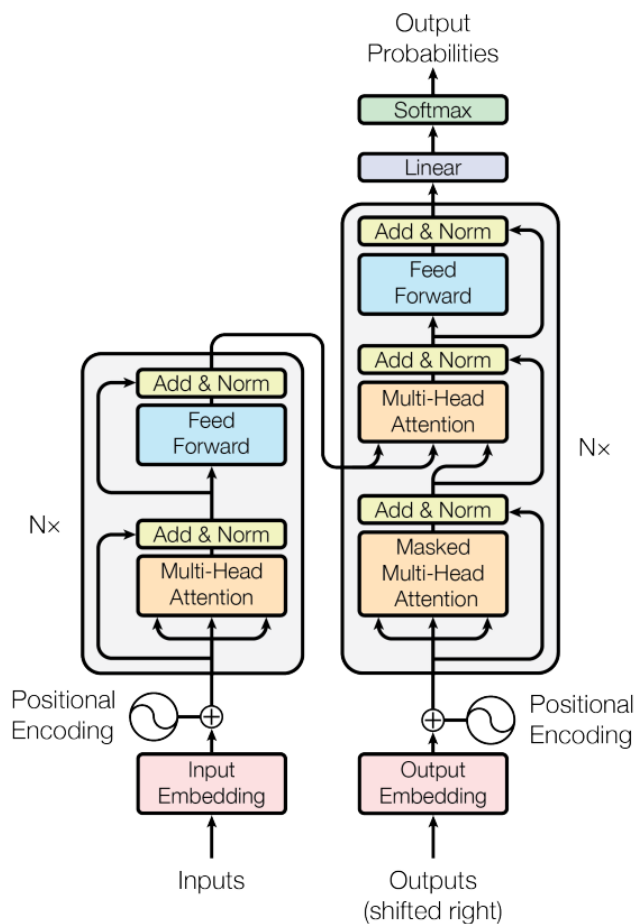
Korišćenjem *self-attention* i *multi-head attention* mehanizama, transformeri prevazilaze ograničenja **RNN**-ova i **LSTM**-ova, pružajući značajna poboljšanja u performansama i efikasnosti za širok spektar **NLP** zadataka.

## 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 dekodeer generiše odgovarajuću izlaznu sekvencu.



Slika 10: Transformer arhitektura [9]

### 2.3.1 Enkoder

Enkoder je odgovoran za obradu ulazne sekvence i ekstrakovanje njenih reprezentacija.

#### Input Embedding

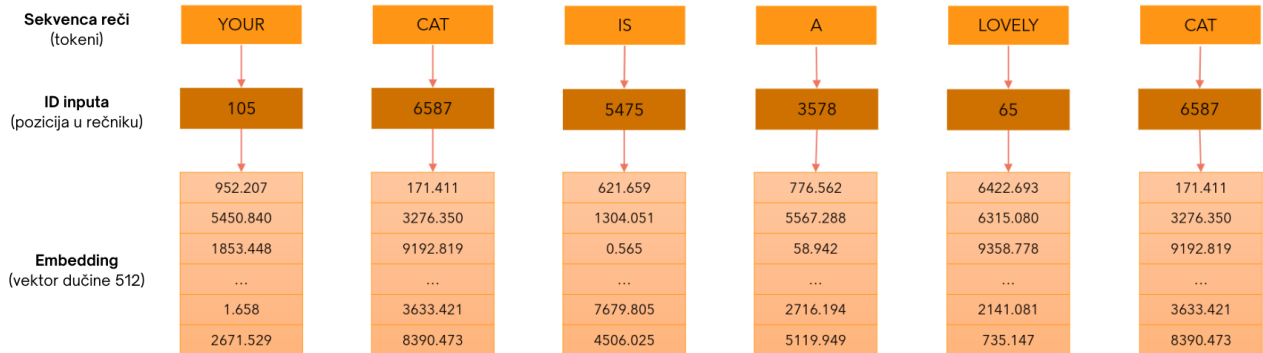
Pre nego što se bilo koja ulazna sekvenca može procesovati od strane transformera, ona mora biti konvertovana u kontinualnu vektorsku reprezentaciju. Ovo se postiže kroz *embedding* sloj, koji mapira svaki token ulazne sekvence u vektor fiksne veličine, označen kao  $d_{\text{model}}$ .

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

$$E : \{x_1, x_2, \dots, x_{\text{seq}}\} \rightarrow \mathbb{R}^{d_{\text{model}}}$$

Tako se ulazna sekvencu  $\{x_1, x_2, \dots, x_{\text{seq}}\}$  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

## Poziciono enkodovanje (engl. 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.

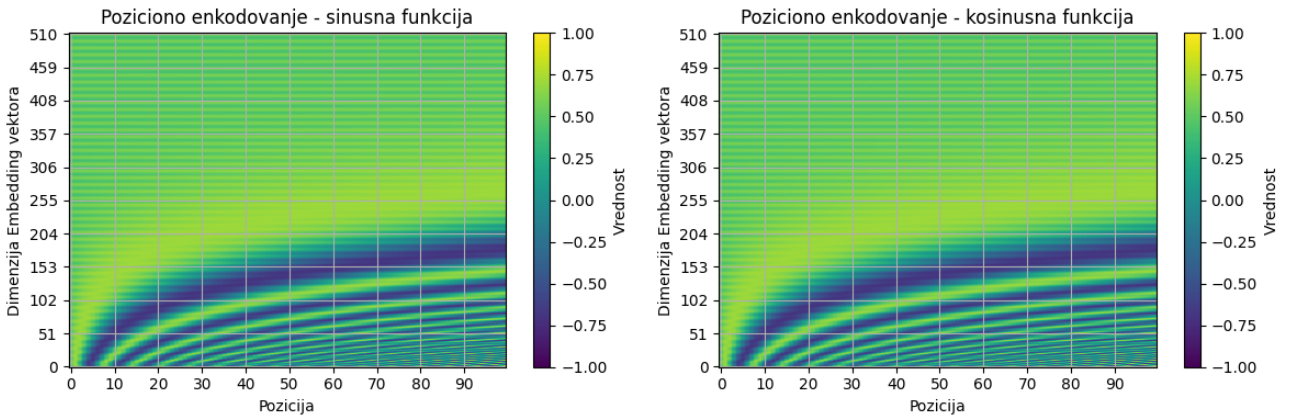
Pozicioni vektori mogu biti naučeni, kao što se uče i *embedding* vektori, ili mogu biti fiksni, zasnovani na unapred definisanim funkcijama. Autori transformera odlučili su da koriste fiksne vektore, koji su davali gotovo iste rezultate kao i naučeni, dok su pritom bili jednostavniji i lakši za interpretaciju.

Fiksni pozicioni vektori za svaku poziciju  $pos$  u sekvenci definirani 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 kontinualni 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 kodovi 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.

### Self-Attention mehanizam

**Self-attention mehanizam** 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. 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**, **Ključ** i **Vrednosti**:

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

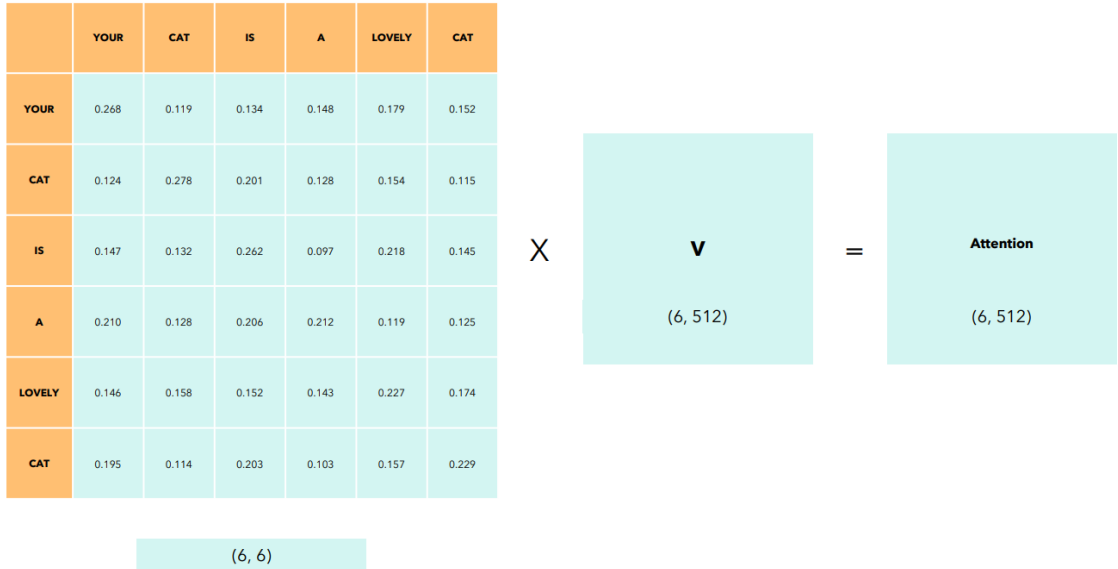
gde su:

- $X$  matrica ulaznih *embedding* vektora dimenzija  $\text{seq} \times d_{\text{model}}$ ,
- $W^Q, W^K, W^V$  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**/**Ključ** 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).

Vrednosti pažnje za svaki par tokena izračunavaju se kao skalarni proizvod **Upita** trenutnog tokena sa **Ključem** drugog tokena, skaliran kvadratnim korenom dimenzije  $d_k$ , nakon čega sledi *softmax* funkcija koja normalizuje rezultate:

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

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 [14]

## Multi-Head Attention

Kako bi se uhvatili 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 podprostorova (glava), od kojih svaka ima svoj skup matrica težina (*Slika 14*). Ovo omogućava modelu da nauči različite aspekte odnosa između tokena.

Za svaku glavu  $i$ :

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

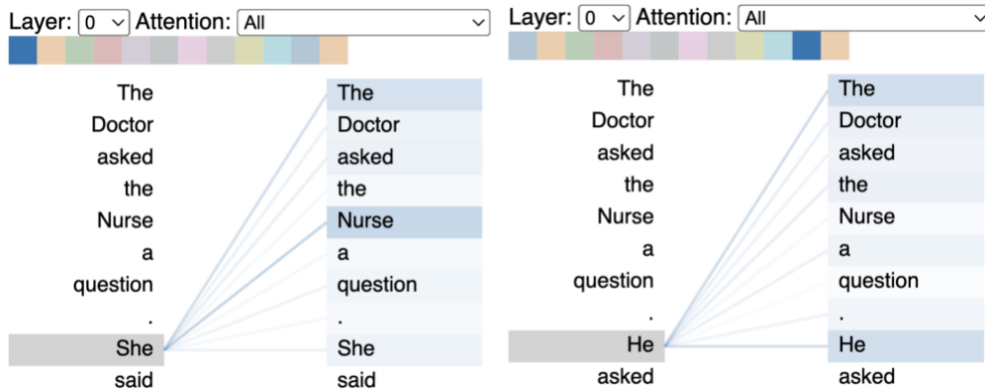
Svaka glava izračunava svoj izlaz pažnje:

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

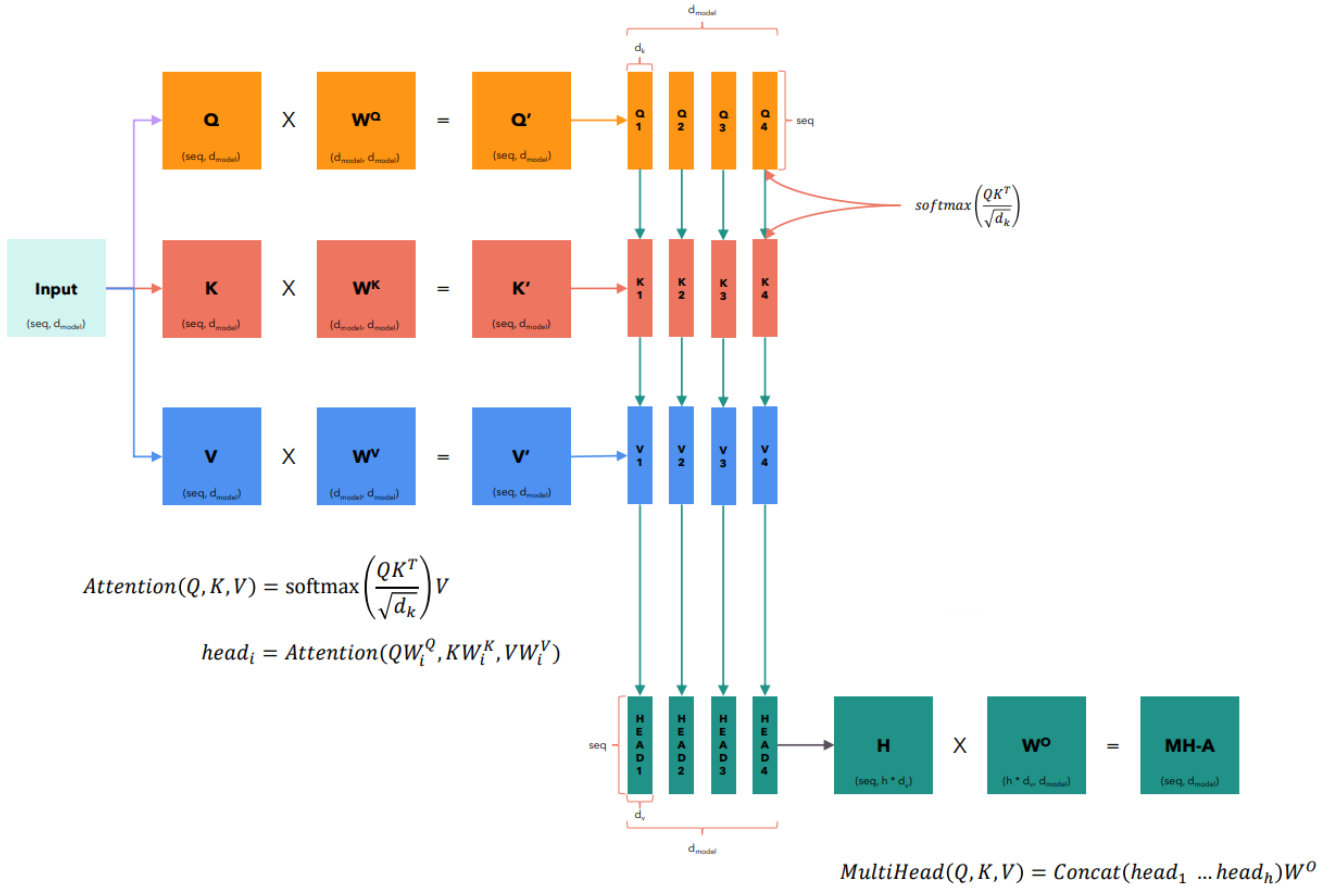
Izlazi svih glava se zatim konkatenuiraju i projektuju nazad na originalnu dimenziju  $d_{\text{model}}$ :

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

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



Slika 14: Vizuelizacija jedne glave *multi-head attention* mehanizma [15]

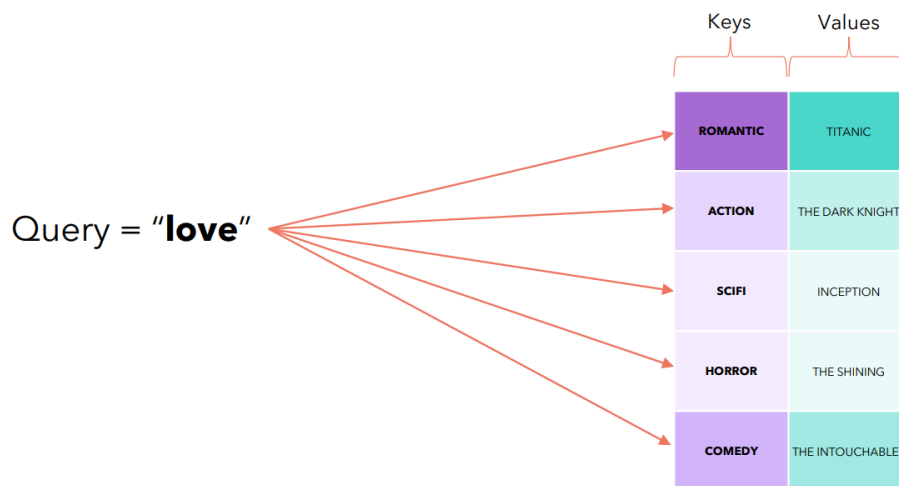


Slika 15: Grafik *multi-head attention* mehanizma [14]

## Intuicija i Uticaj

Notacija  $K$ ,  $Q$  i  $V$  inspirisana je konceptima ključ (key), upita (query) i vrednosti (value) iz sistema bazi podataka (Slika 16).

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



Slika 16: Koncepti ključa, upita i vrednosti vuku inspiraciju u bazama podataka [14]

## 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.

Formalno:

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

gde:

- $x$  je ulazni vektor.
- $\mu$  i  $\sigma$  su srednja vrednost i standardna devijacija  $x$ .
- $\gamma$  i  $\beta$  su parametri za skaliranje i pomeranje koje 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 može koristiti dublje arhitekture mreže 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.

## 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 query, key, i value matrice.
- $d_k$  je dimenzija ključeva (keys).
- $M$  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

## Enkoder-Dekoder Attention

Nakon računanja maskiranog *self-attention* mehanizma, dekodeer obraća pažnju na izlaz enkodera koristeći standardni *attention* mehanizam. Ovo omogućava dekodeeru 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 dekodeera, dok  $K$  i  $V$  dolaze iz završnih slojeva enkodera.

## Feed-Forward Neuronska Mreža (FFN)

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

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

gde:

- $W_1, W_2$  su težinske matrice, a  $b_1, b_2$  su pomerači.
- *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.

## 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  $n$ , jer model obrađuje sve pozicije do trenutnog tokena:

- **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*" [16] autora Dosovitskiy i ostalih saradnika (2020). Primarna inovacija **ViT**-a je prilagođavanje arhitekture transformera, prvobitno dizajnirane **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 ovu ugrađenu prostornu induktivnost. Umesto toga, uči odnose između svih delova slike putem *self-attention* mehanizma, što im omogućava da modeluju globalne zavisnosti i interakcije između različitih delova slike.

- **Receptivno Polje:**

- **CNN:** Ima ograničeno receptivno polje, što znači da svaki filter 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.

### 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.
- **Skalabilnost:** Mogu se efikasno skalirati povećanjem podataka i računarskih resursa, koristeći prethodno obučene velike modele.
- **Fleksibilnost:** Nisu ograničeni na lokalnu obradu kao **CNN**-ovi i mogu da nauče reprezentacije koje nisu ograničene strukturom konvolucionih filtara.

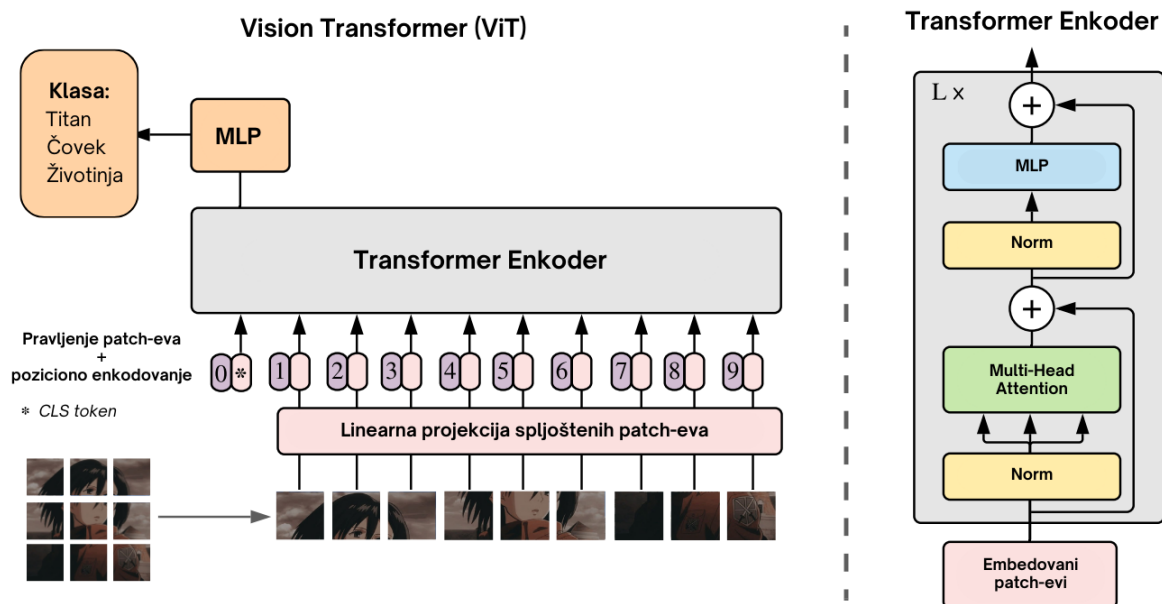
- **Nedostaci:**

- **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.
- **Računski Trošak:** Kvadratna složenost *self-attention* mehanizma može biti računski skupa, posebno za slike visoke rezolucije ili duge nizove tokene.
- **Složenost Obuke:** Obuka može biti izazovna i često zahteva pažljivo podešavanje hiperparametara i velike računarske resurse.

### 3.4 Arhitektura Vision Transformera

Glavni delovi ViT arhitekture su:

- Tokenizacija slike
- Poziciono enkodovanje
- Transformer Encoder
- Klasifikacioni MLP<sup>2</sup>



Slika 18: Arhitektura Vision Transformera za klasifikaciju slika

<sup>2</sup>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.

#### Deljenje slike na pečeve

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  $P \times P$ .

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

$$N = \frac{H \times W}{P^2}$$

Svaki *patch* se zatim "spljošti" u jednodimenzionalni vektor veličine  $P^2 \times C$ .

#### Embedovanje 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 embedding sloj (engl. *patch embedding layer*).

Za dati spljošteni *patch*  $x_i \in \mathbb{R}^{P^2 \times C}$ , embedding 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  $W_e \in \mathbb{R}^{(P^2 \times C) \times d_{model}}$  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 enkodovanja, koji se sabira sa ovim vektorima pre nego što prođu kroz enkoder.

### 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.

#### Fiksni pozicioni vektori

U **ViT** modelu, pozicioni vektori se često generiše 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 [16], pa se koristi jednodimenzionalni pristup.

## Pozicioni vektori koji se uči

Umesto korišćenja fiksnih sinusnih i kosinusnih funkcija, moguće je i da se pozicioni vektori nauči 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 jednostavnija za implementaciju i računanje.

### 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 se sastoji od više slojeva, od kojih svaki uključuje:

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

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

### 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. Konkretna reprezentacija ovog tokena, nakon prolaska kroz enkoder, koristi se za klasifikaciju.

Upotreba [CLS] tokena u ViT-u je inspirisana modelom BERT [12] 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.

## 4 Eksperimenti i Rezultati

### 4.1 Postavka eksperimenta

#### 4.1.1 Softver

Eksperimenti su vršeni koristeći sledeći softver:

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



#### 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<sup>3</sup> 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 [17].

#### 4.1.3 Dataset

Eksperimenti su sprovedeni koristeći MNIST dataset [18], 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.

---

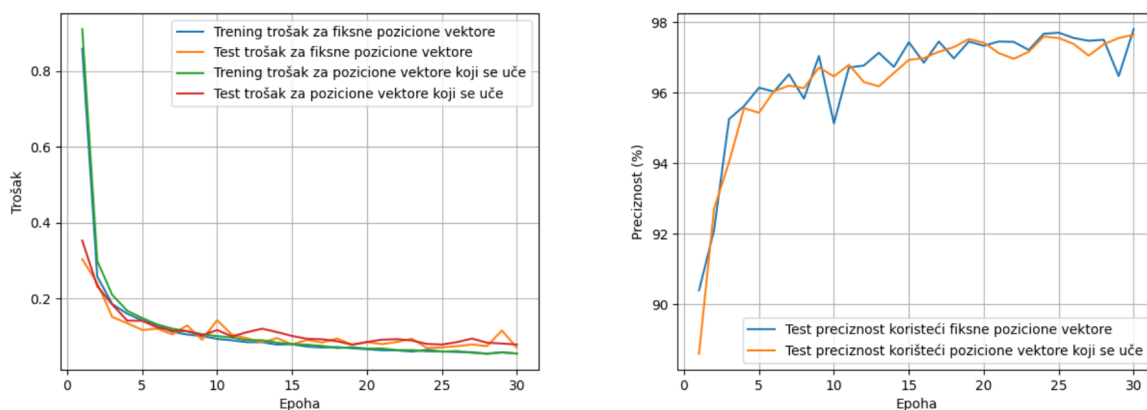
<sup>3</sup>Kod je dostupan na [https://github.com/Kovelja009/ViT\\_implementation](https://github.com/Kovelja009/ViT_implementation)

## 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.

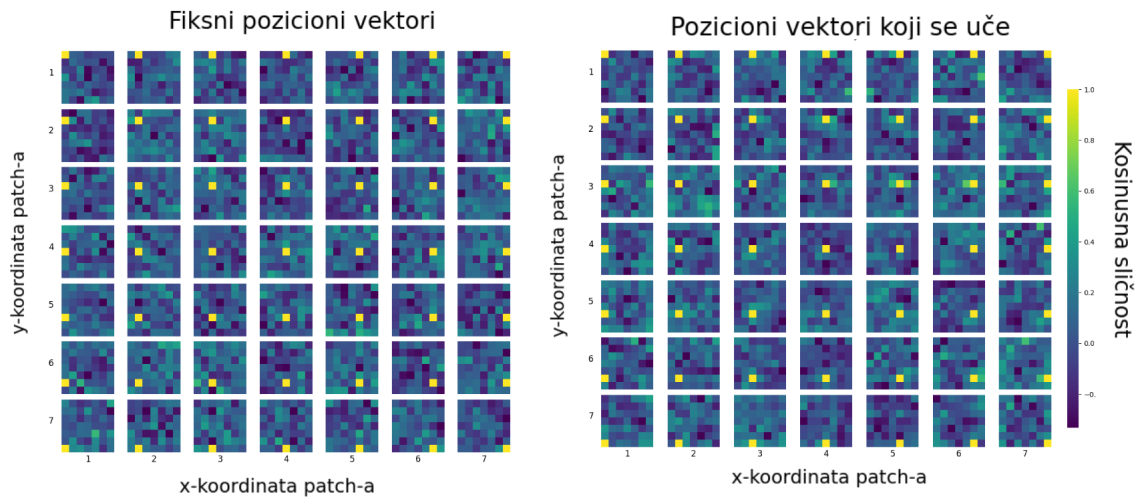
Rezultati:

- **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.



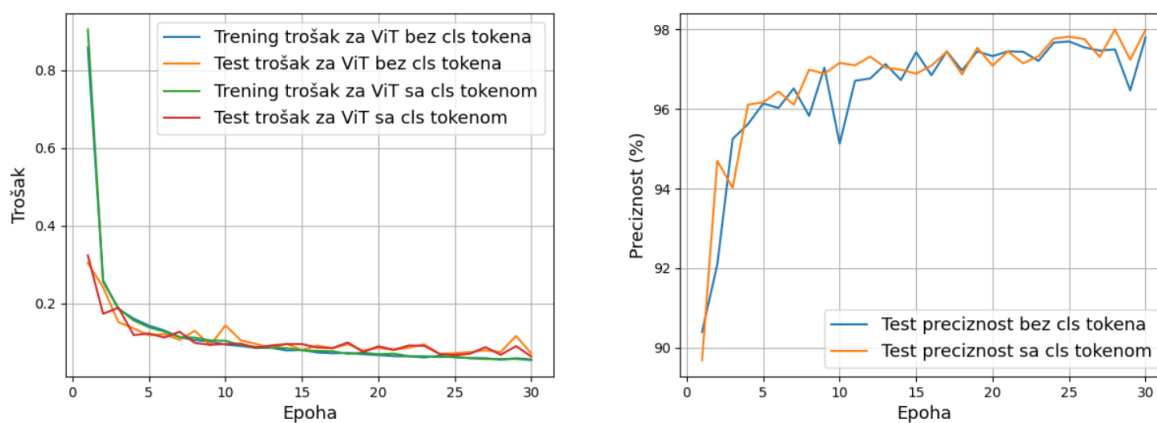
Slika 20: Kosinusna sličnost između različitih *patch*-eva za fiksne i pozicione vektore koji se uče

### 4.3 Eksperiment 2: Uticaj uklanjanja CLS tokena

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

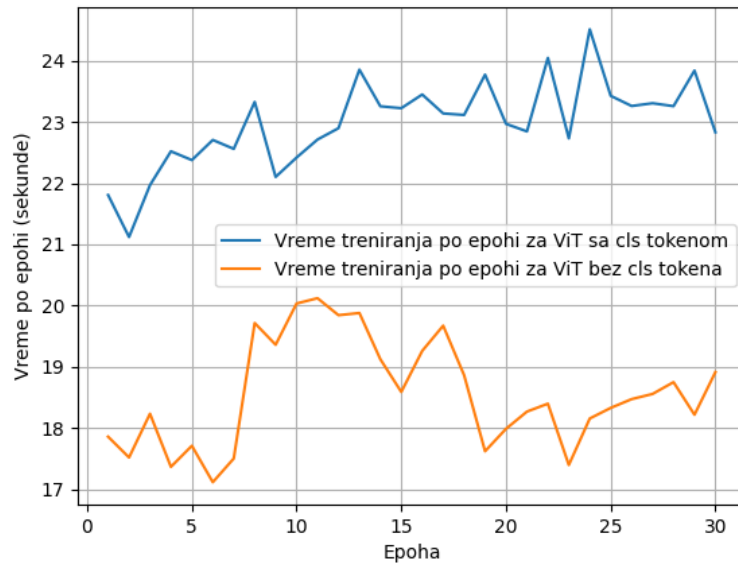
#### Rezultati:

- **Performanse Modela:** *Slika 21* potvrđuje da ViT ima skoro identične performanse sa i bez CLS tokena, što je u skladu sa hipotezom prikazanom u originalnom radu [16].



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 klasa token izostavi (približno 1.3 puta brži). Pretpostavka je da to zbog načina na koji je CLS token implementiran i ostaje da se dalje istraži.



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