Универзитет у Нишу

Електронски факултет

Катедра за рачунарство

Мастер рад

Архитектуре визуелних трансформера за класификацију слика

Стеван Грујић, 1493

Ниш, 2024.

Универзитет у Нишу

Електронски факултет

Катедра за рачунарство

Мастер рад

Архитектуре визуелних трансформера за класификацију слика

Задатак: Истражити архитектуре визуелних трансформера за класификацију слика. Извршити поређење различитих трансформера са конволуционим неуронским мрежама и темељно анализирати архитектуре и начин рада визуелних трансформера.

Ментор:

Проф. др Александар Милосављевић

Кандидат:

Стеван Грујић 1493

Комисија:

1._____

2._____

3._____

Датум пријаве: _____

Датум предаје: _____

Датум одбране: _____

АРХИТЕКТУРЕ ВИЗУЕЛНИХ ТРАНСФОРМЕРА ЗА КЛАСИФИКАЦИЈУ СЛИКА

САЖЕТАК

У данашњем дигиталном добу, са већим обимом и разноликошћу визуелних података, потреба за ефикасним методама класификације слика постаје кључна за различите индустрије и апликације. У раду се прво дубински анализира еволуција и примена различитих метода класификације слика, пружајући широк увид у ову област и истовремено истичући изазове и ограничења досадашњих приступа. Након увода у основне концепте, фокус се премешта на појам визуелних трансформера, наглашавајући бенефите употребе овог приступа у класификацији слика. Затим се детаљно разматра иницијална архитектура визуелних трансформера, пореди се са трансформер архитектурама које су претежно коришћене у обради природних језика, а посебан нагласак ставља се на механизам самопажње који чини срж ових архитектура. Иако иницијални визуелни трансформери теже да замене конволуционе неуронске мреже, суочавају се са изазовима у перформансама, што доводи до потребе за даљим оптимизацијама.

У том контексту, рад детаљно описује нове трансформер архитектуре, међу којима се истичу *DeiT* и *Swin*. *DeiT* се ослања на принцип дестилације знања како би смањила комплексност архитектуре, док *Swin* проширује примену визуелних трансформера на шири спектар задатака рачунарског вида, укључујући детекцију објеката и семантичку анализу. Кроз опис ових нових приступа, истраживање истиче потенцијалне иновације које доносе у области класификације слика.

Главни циљ истраживања јесте темељна анализа поменутих архитектура и њихово поређење кроз експерименте над скупом података о воденим инсектима. У практичном делу рада, *ViT*, *DeiT* и *Swin* се примењују кроз технику пренесеног учења како би се постигли оптимални резултати класификације. Такође, извршена је и имплементација *ViT* архитектуре од нуле како би се додатно продубило разумевање ових концепта. Кроз ове експерименте, рад истражује како теоријске предности нових архитектура утичу на њихову практичну примену.

Коначно, рад обухвата и детаљну анализу резултата експеримената, на тај начин истичући и концепт индуктивне пристраности као кључни фактор за добијене резултате. Ова анализа пружа дубљи увид у перформансе различитих архитектура и њихову способност генерализације на различите скупове података. Кроз све ове кораке, рад доприноси разумевању и унапређењу метода класификације слика кроз примену визуелних трансформера, отварајући пут ка новим иновацијама и напретку у области рачунарског вида.

Кључне речи: визуелни трансформер, трансформер енкодер, механизам самопажње, дестилација знања, индуктивна пристрасност

VISION TRANSFORMER ARCHITECTURES FOR IMAGE CLASSIFICATION

ABSTRACT

In today's digital age, with a larger volume and diversity of visual data, the need for efficient image classification methods becomes crucial for various industries and applications. The paper first deeply analyzes the evolution and application of various image classification methods, providing a broad insight into this field while highlighting the challenges and limitations of previous approaches. After introducing the basic concepts, the focus shifts to the concept of vision transformers, emphasizing the benefits of using this approach in image classification. It then thoroughly discusses the initial architecture of vision transformers, comparing it with transformer architectures mainly used in natural language processing, with a special emphasis on the self-attention mechanism that forms the core of these architectures. Although initial vision transformers aim to replace convolutional neural networks, they face challenges in performance, leading to the need for further optimizations.

In this context, the paper extensively describes new transformer architectures, among which DeiT and Swin stand out. DeiT relies on the principle of knowledge distillation to reduce the complexity of the architecture, while Swin extends the application of vision transformers to a wider range of computer vision tasks, including object detection and semantic analysis. Through the description of these new approaches, the research highlights the potential innovations they bring to the field of image classification.

The main goal of the research is a thorough analysis of the mentioned architectures and their comparison through experiments on a dataset of aquatic insects. In the practical part of the work, ViT, DeiT, and Swin are applied through transfer learning techniques to achieve optimal classification results. Additionally, the implementation of the ViT architecture from scratch is carried out to further deepen the understanding of these concepts. Through these experiments, the paper explores how the theoretical advantages of new architectures impact their practical application.

Finally, the paper also includes a detailed analysis of the experimental results, thus highlighting the concept of inductive bias as a key factor for the obtained results. This analysis provides a deeper insight into the performance of different architectures and their ability to generalize to different datasets. Through all these steps, the paper contributes to understanding and improving methods of image classification through the application of vision transformers, paving the way for new innovations and advancements in the field of computer vision.

Keywords: vision transformer, transformer encoder, self-attention mechanism, knowledge distillation, inductive bias

Садржај

1 Увод

Са експанзијом дигиталних технологија и убрзаним развојем рачунарских система, анализа и класификација слика постали су неизоставни елементи различитих домена. Само неки од споменутих домена су: медицинска дијагностика, аутономна возила, контрола квалитета промета, и др. Еволуција у домену класификације слика сведочи о континуираном напретку, у којем се неуронске мреже истичу као кључни играчи у постизању изузетних перформанси у различитим задацима. Од пионирских радова током 1950-их до процвата дубоког учења и појављивања моћних архитектура попут *AlexNet* и *VGGNet*, ова еволуција довела је до фасцинантних резултата у домену рачунарског вида.

У међувремену, трансформерске архитектуре, које су првобитно развијене за обраду природног језика, стекле су огромну популарност и примену у различитим доменима, укључујући и класификацију слика. Основна идеја трансформер архитектуре, са својим механизмом само-пажње (енг. *self-attention mechanism*), показала се изузетно ефикасном у хватању глобалних зависности у подацима. Ова адаптација трансформерских архитектура на анализу слика представља велики корак напред и отвара врата новим могућностима у домену рачунарског вида.

Овај рад има за циљ да пружи преглед развоја класификације слика, истражити основне концепте трансформерских архитектура, као и анализирати специфичне приступе неких од популарних трансформера попут, *ViT*-а (*VisionTransformer*), *DeiT*-а (*Data-efficient image Transformer)* и *Swin* (*Shifted Window*) трансформера. Кроз детаљно објашњење сваког појединачног блока трансформер архитектуре, као и анализу перформанси нових архитектура, рад ће истражити како се трансформери интегришу у област рачунарског вида и како унапређују класификацију слика. Кроз ово истраживање, циљ је пружити дубоко разумевање примене трансформерских архитектура у класификацији слика, разоткривајући њихов потенцијал за унапређење перформанси и решавање изазова у овој динамичној области рачунарског вида.

2 Развој метода за класификацију слика

У овом поглављу, детаљно ће бити истражена еволуција класификације слика, фокусирајући се на различите архитектуре неуронских мрежа које су допринеле напретку у овој области. Почевши од раних периода када су вештачке неуронске мреже биле у својим првим корацима, пратићемо њихов развој кроз важне архитектуре као што су *LeNet*-5 и *AlexNet*. За сваку од ових архитектура, укратко ће бити размотрене кључне иновације, предности и ограничења која су истакнута у њиховом времену.

Након тога, биће анализирани новији модели, укључујући *VGGNet* и трансформер архитектуру. Поред тога, детаљније ће бити објашњено како су ови модели допринели еволуцији концепта класификације слика. Посебна пажња биће посвећена трансформерима, који су увели револуционарни приступ обради слика користећи механизам само-пажње (енг. *self-attention*). Кроз ово поглавље, читаоци ће добити увид у трансформативне промене у области класификације слика и разумети значај и утицај различитих неуронских мрежа на ову дисциплину.

2.1 Вештачке неуронске мреже

Увођење вештачких неуронских мрежа (енг. *Artificial Neural Network*) током 1950-их година представљало је револуционарни корак у развоју вештачке интелигенције. Пионирски радови попут перцептрона, представљеног од стране Франка Росенблата касних 1950-их, поставили су темеље за разумевање неуронских мрежа. Међутим, тадашња ограничења рачунарских ресурса нису дозвољавала практичну примену ових мрежа. Због овога, уследио је период у коме је развој вештачке интелигенције био веома скроман.

У наредним деценијама, посебно у 1980-им годинама, долази до обнове интересовања за неуронске мреже, захваљујући открићу алгоритма који носи назив пропагација уназад (енг. *back-propagation*). Овај алгоритам омогућава ефикасно тренирање вишеслојних неуронских мрежа, чиме се, на одређени начин, превазилазе претходне препреке. Овај период обележен је оптимизацијом и усавршавањем техника тренирања, што је допринело поновном порасту интереса за неуронске мреже. [1]

Међутим, упркос напретку, изазови су остали присутни. Нестајући градијенти, који се често јављају током тренирања дубоких мрежа, представљали су озбиљан проблем у класификацији слика. Такође, опасност од преприлагођавања (енг. *overfitting*) била је и даље актуелна, чиме су перформансе неуронских мрежа на комплексним задацима биле ограничене.

Овај период еволуције неуронских мрежа припремио је терен за следеће кључне кораке у развоју технологије, укључујући интеграцију концепта неуронских мрежа у област класификације слика, што ће се детаљније истражити у наставку рада.

2.2 Генеза и развој конволуционих неуронских мрежа

Конволуционе неуронске мреже (енг. *Convolutional Neural Networks, CNNs*) произишле су из потребе за ефикасним решењем задатка класификације ручно писаних цифара у оквиру *MNIST* скупа података у области рачунарског вида. Иако би људима ово можда изгледало као једноставан задатак, за моделе рачунарске интелигенције тог времена представљао је озбиљан изазов. Иницијално су се користиле *feedforward* неуронске мреже (енг. *Feedforward Neural Networks, FFNNs*) или вишеслојни перцептрони (енг. *Multi Layer Perceptron, MLPs*) за решавање овог проблема. Међутим, *feedforward* неуронске мреже, са својим густо повезаним слојевима, нису могли постићи високу тачност због немогућности хватања просторних хијерархија и високе димензионалности сликовних података. [2] Наиме, модели засновани на вишеслојним перцептронима су били превише генерализовани за један овакав задатак. Разлог овоме је веома мали *inductive bias*, тј. непостојање било каквог усмеравања модела у правцу извршавања конкретног задатка (што је касније био случај са појавом конволуционих неуронских мрежа). Примера ради, померање слике улево или удесно за само један пиксел довело би до добијања лоших резултата.

2.2.1 *LeNet*-5

Почетком 1990-их година, појављује се једно револуционарно решење француско-америчког научника *Yann LeCun-a* који је представио идеју конволуционих неуронских мрежа како би превазишао горенаведене изазове. У свом научном раду, *LeCun* је представио архитектуру под називом ***LeNet*-5**, која је користила конволуционе слојеве како би разумела хијерархијске обрасце у подацима, што је било кључно одступање од густо повезаних слојева. Ова иновативна архитектура, поставила је темеље за еволуцију конволуционих неуронских мрежа и имала дубок утицај на поље рачунарског вида уопште. [2]

LeNet-5 се састојао од неколико кључних елемената који су унапредили његову ефикасност у решавању проблема класификације руком писаних цифара:

1) **Локална перцептивна поља (*Local Receptive Fields*):** Уместо да сваки неурон у слоју буде повезан са сваким неуроном у претходном слоју, као што је то раније био случај, *LeNet*-5 је увео концепт локалних перцептивних поља. Ово значи да је сваки неурон у одређеном слоју био повезан само са малим подручјем претходног слоја. Ова локалност омогућила је мрежи да учи локалне карактеристике, смањујући број параметара и олакшавајући генерализацију на различите задатке.

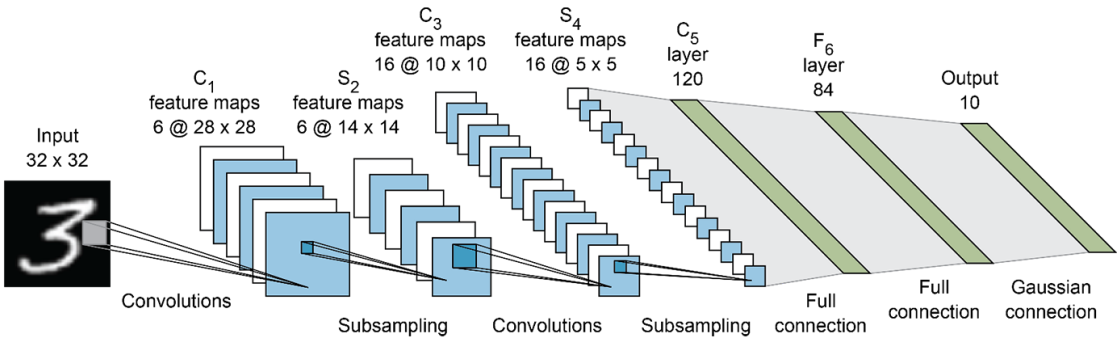
2) **Дељење тежина**: Сваки неурон у одређеној мапи карактеристика (енг. *feature map*) делио је исти скуп тежина са свим другим неуронима у тој мапи. Ова техника, позната као тежинско дељење, допринела је смањењу укупног броја параметара у мрежи, чинећи је ефикаснијом и мање склоној преприлагођавању.

3) ***Subsampling***: *LeNet*-5 је користио *subsampling* слојеве, попут *max-pooling*-а, како би смањио просторну димензију мапа карактеристика. Ова техника је доприносила

смањењу рачунарског оптерећења и повећању ефикасности. Такође, овај приступ омогућио је сузбијање, до тада веома присутног, преприлагођавања.

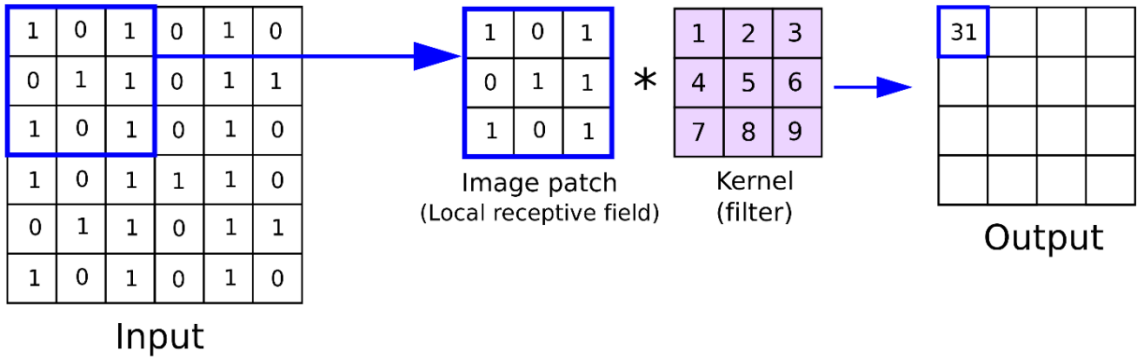
4) **Конволуциони слојеви**: Централна компонента *LeNet*-5 архитектуре били су конволуциони слојеви. Коришћење ових слојева омогућило је мрежи да научи просторно инваријантне карактеристике, посебно важне за задатке као што је препознавање руком писаних цифара, где позиција цифре на слици може варирати.

LeNet архитектура се састоји од два конволуциона слоја, два *subsampling* слоја (*max-pooling* слоја) и три потпуно повезана слоја. Укратко, *LeNet* архитектура показује снагу конволуционих неуронских мрежа у задацима препознавања слика и поставља темеље за многе касније развоје у алгоритмима дубоког учења.



Слика 2.2.1.1 *LeNet*-5 архитектура

Конволуциони слојеви су састављени од конволуционих филтера (кернела). Ови филтери се користе за различите задатке у области рачунарског вида, изоштравање и замућивање слика, детекција ивица, и др. Величина кернела је најчешће 3x3, а сам поступак примене ових филтера се назива конволуција.



Слика 2.2.1.2 Конволуциони филтер (кернел)

Илустроваћемо начин рада конволуције на примеру. Узимимо у обзир улазну слику висине *H*, ширине *W* и броја канала *C*. Поред овога, имамо и скуп од *K* кернела, при чему сваки од њих, такође, има своју ширину, висину и број канала. Број канала је у овом

случају, углавном, много већи у поређењу са улазном сликом, док су, са друге стране, висина и ширина неупоредиво мање у односу на резолуцију улазне слике.

2*D* операција конволуције узима сваки филтер посебно и њиме „клизи“ преко слике. На свакој позицији, рачуна *element-wise* производ између филтера и одговарајућих пиксела улазне слике, а потом врши суму резултујућих вредности како би се произвела јединствена вредност на излазу. Важно је напоменути да вредности пиксела улазне слике нису директно повезане са излазним слојем у делимично повезаном слоју.

Да бисмо израчунали излазну мапу карактеристика за све филтере, потребно је извршити конволуциону операцију К пута, једном за сваки филтер. Излазна мапа се затим добија стапањем добијених мапа дуж осе дубине. Важно је напоменути да се величина излазне мапе карактеристика одређује величином улазне слике, величином филтера, кораком и *padding*-ом.

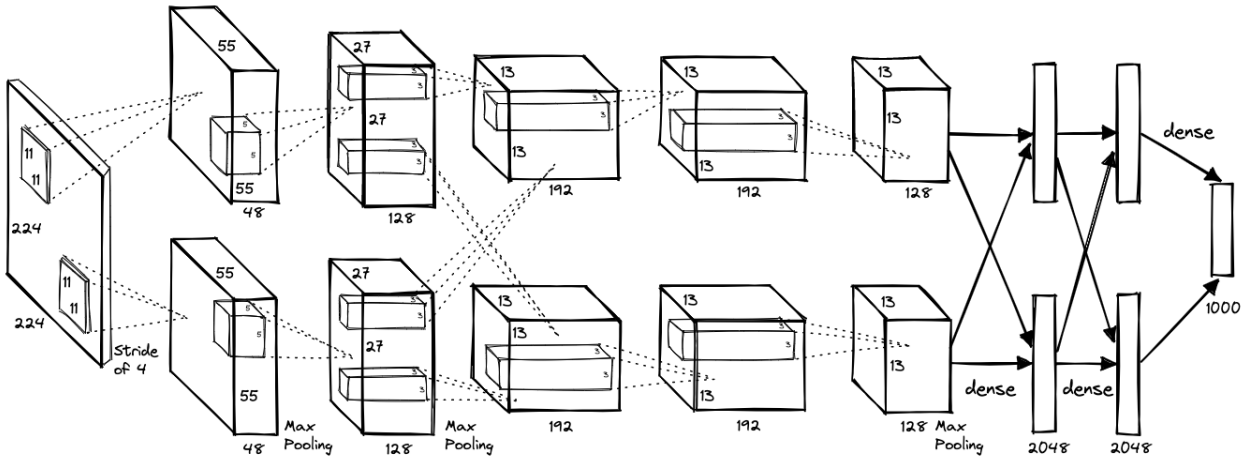
2.2.2 AlexNet

Захваљујући чињеници да је *LeNet* постигао тачност од 99% на задатку класификације ручно писаних цифара у оквиру *MNIST* скупа података, није остало пуно простора за побољшање. Због овога је научна заједница увидела потребу за неким изазовнијим задатком како би, на тај начин, подстакла даљи развој модела.

ImageNet изазов, познат и као *ImageNet Large Scale Visual Recognition Challenge (*скр. *ILSVRC*), представљао је управо такав изазов какав је научна заједница тражила. Овај такмичарски формат, одржаван од 2010. до 2017. године, играо је кључну улогу у напредовању конволуционих неуронских мрежа.

У поређењу са *MNIST*-ом, *ImageNet* скуп података садржи значајно веће количине *RGB* слика - око 1 милион, распоређених у 1000 различитих класа. Слике имају већу резолуцију, на тај начин омогућавајући већу количину детаља. Ова разноврсност чини *ImageNet* скуп података широко прихваћеним скупом за претренирање (енг. *pretraining*) готово свих задатака рачунарског вида.

Године 2012, модел по имену ***AlexNet***, који је дизајниран помоћу конволуционих неуронских мрежа, освојио је прву награду на овом изазову са значајном предношћу у односу на другопласирани модел, на тај начин означавајући парадигмски помак у области рачунарског вида.



Слика 2.2.2.1 *AlexNet* архитектура

Кључни разлози за успех *AlexNet* -а били су:

1) **Велики скуп података**: *AlexNet* је трениран на великом скупу слика под именом *ImageNet*, који је садржао преко 1,2 милиона тренинг слика и 50.000 валидационих слика распоређених у 1000 различитих класа. Овај велики скуп података омогућио је мрежи да научи широк спектар особина и образаца корисних за класификацију слика.

2) **Дубока архитектура**: *AlexNet* је био дубока архитектура са осам слојева, укључујући пет конволуционих слојева и три потпуно повезана слоја. Коришћење више слојева омогућило је мрежи да научи сложене хијерархијске репрезентације улазних слика, што је било кључно за тачну класификацију.

3) **Конволуциони слојеви**: Коришћење конволуционих слојева омогућило је *AlexNet* -у да учи просторно инваријантне особине корисне за класификацију слика. Кроз слој за узорковање, научили су скуп филтера који се конволутују са улазном сликом, производећи скуп мапа особина које бележе различите аспекте слике.

4) **Активација *ReLU***: *AlexNet* је користио *ReLU* активациону функцију, што је омогућило мрежи брже учење у поређењу са претходним архитектурама које су махом користиле сигмоидалне или тангенс активације. *ReLU* такође помаже у спречавању проблема нестајућег градијента који се може јавити у дубоким неуронским мрежама.

5) **Аугментација података**: *AlexNet* је користио технике аугментације података као што су сечење, окретање и скалирање како би вештачки повећао величину тренинг

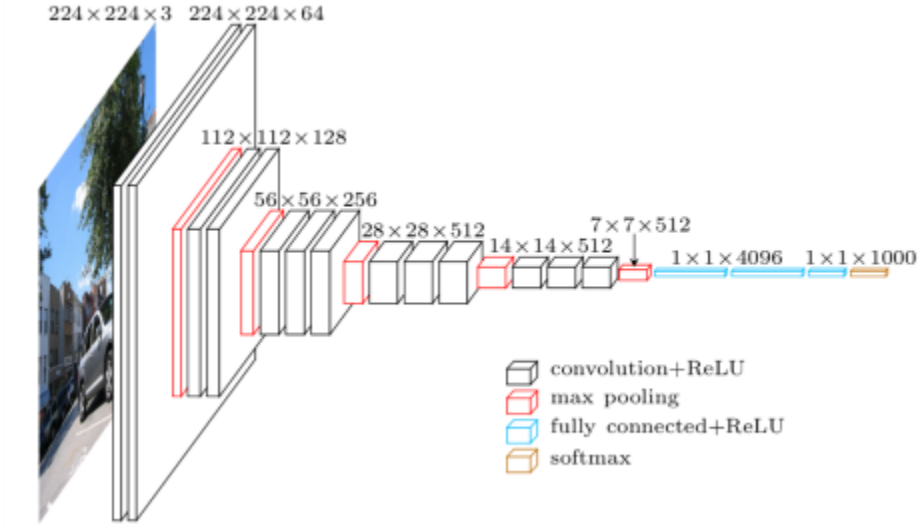
скупа података. Ово је помогло смањењу преприлагођавања и побољшало генерализацију перформанси мреже.

6) ***Dropout* регуларизација**: *AlexNet* је користио *dropout* регуларизацију, која је случајно искључивала одређени проценат неурона током тренирања како би спречила преприлагођавање. Ова техника такође помаже у побољшању генерализације перформанси мреже.

2.2.3 Конволуционе мреже новијег доба

Након *AlexNet*-а, развијено је и фино подешено (енг. *fine-tuned)* више модела базираних на конволуцијама, који су служили као моћни екстрактори карактеристика (енг. *feature*) за различите задатке изван класификације слика, укључујући детекцију објеката, сегментацију и друге.

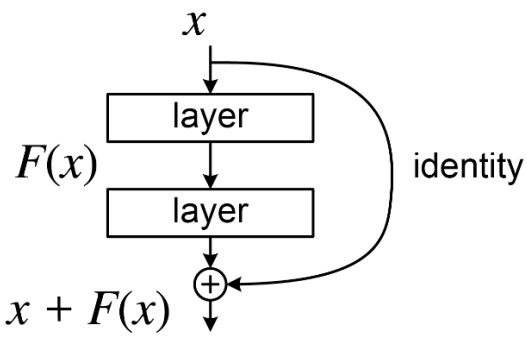
Свакако је потребно споменути и ***VGGNet***, или једноставно *VGG*, који је предложен од стране *Visual Geometry Group* на Универзитету у Оксфорду. *VGG* је био још један кључни корак у еволуцији конволуционих неуронских мрежа, будући да је био први модел који је широко коришћен као основна архитектура захваљујући својој униформној структури, што га чини високо прилагодљивим за различите задатке. *VGG* је показао моћ дубине у неуронским мрежама, користећи између 16 и 19 слојева и користећи мале (3x3) конволуционе филтере широм читаве неуронске мреже.



Слика 2.2.3.1 *VGGNet* архитектура

И поред моћи дубине коју је *VGG* показао, тренирање веома дубоких мрежа имало је један озбиљан проблем, а то су нестајући градијенти. Са порастом дубине мреже, градијенти добијени у процесу пропагације уназад често постајали изузетно мали, на тај начин спречавајући мрежу да учи. *ResNet* (*Residual Network)*, представљен од стране

Microsoft Research тима, решио је овај проблем уводећи прескачуће везе (енг. *skip-connections*) (Више о овоме у каснијим поглављима) који су омогућавали градијенатима да буду директно враћени преко пропагације уназад ка ранијим слојевима. Ова архитектура омогућила је тренирање мрежа које су значајно дубље него икада пре, а тим је представио модел са 152 слоја за *ImageNet* такмичење. Ово је био значајан корак напред, с обзиром на то да је показао да се мреже произвољне дубине могу ефикасно тренирати и постизати боље перформансе од плићих модела.



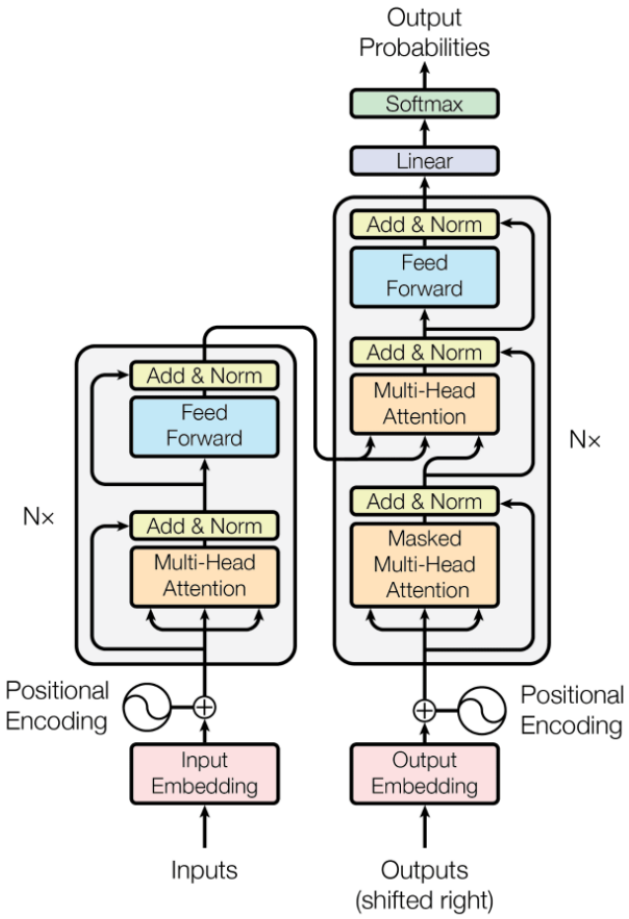
Слика 2.2.3.2 Резидуални блок

2.3 Појава трансформер архитектуре

2.3.1 Оригинална трансформер архитектура

Конволуционе неуронске мреже имају низ кључних предности које су допринеле њиховој опсежној употреби и успеху. Оне инхерентно могу хватати просторне хијерархије, и због тога су се до сада сматрале идеалним за задатке обраде слика. Штавише, њихове архитектуре, од раних *LeNet* до новијих *ConvNet*, непрестано су еволуирале како би побољшале екстракцију карактеристика, носиле се са дубљим мрежама и ефикасно рачунале с мање података.

Међутим, средином 2017. године *Google* је представио свој револуционарни рад [3] у којем је представљена, иновативна архитектура, под називом **трансформер архитектура.** Ускоро је овај приступ преузео водећу улогу, пре свега у домену обраде природних језика, а затим и у другим областима. Као резултат тога, потпуно је ставио у сенку, до тада незаменљиве мреже дугорочне и краткорочне меморије (*LSTM*), постављајући нови стандард у ефикасности и способности моделирања сложених веза унутар података.



Слика 2.3.1 Трансформер архитектура [3]

Енкодер-декодер архитектура трансформер модела (приказана на слици 2.3.1), састоји се од два главна дела: енкодера и декодера. Оба ова дела користе идентичне блокове, али имају различите улоге у процесу обраде секвенци.

Енкодер има задатак да научи репрезентацију улазне секвенце, попут низа речи у реченици. Сваки блок енкодера састоји се од два кључна слоја: *Multi-Head-Attention* (скр. *MHA*) и *Feed Forward* мрежа. *MHA* омогућава моделу да усмери своју пажњу на различите делове улазне секвенце истовремено, омогућавајући му да ефикасно хвата зависности и односе међу подацима. *Feedforward* мрежа, са друге стране, даље обрађује ове информације. Ова, дуална архитектура, омогућава енкодеру да ефикасно апстрахује и представи информације из улазних секвенци.

Декодер, са друге стране, дели сличну структуру блокова са енкодером, али инкорпорира додатне слојеве с циљем предвиђања наредног дела секвенце. Такође, користи механизам пажње, али у овом контексту, усмерава пажњу ка енкодираним репрезентацијама како би се нагласила релевантност информација за тренутни корак у декодирању. Ова иновативна употреба механизма пажње омогућава декодеру да ефикасно синтетише информације из енкодиране репрезентације, пружајући прецизност и контекстуалну дубину у процесу генерисања секвенци.

Неки од најистакнутијих примера примене трансформер архитектуре, у обради природних језика, обухватају моделе као што су *BERT* (*Bidirectional Encoder Representations from* Transformers) и *GPT* (*Generative Pre-trained Transformer*).

BERT, развијен од стране *Google*-а, се базира на искључивој енкодерској архитектури без укључивања декодера. Током обуке, *BERT* анализира секвенцу текста у оба смера истовремено, омогућавајући моделу да створи дубоко разумевање контекста речи и прецизно репрезентује семантичке односе у тексту.

Супротно томе, *GPT* модели, развијени од стране *OpenAI*-а, су оријентисани ка генеративним способностима и користе искључиво декодерску архитектуру. Ови модели, способни су генерисати текст на основу датог контекста, истичући се као једни од најмоћнијих језичких модела способних за разноврсне језичке задатке, укључујући и смерано генерисање одговора.

2.3.2 Утицај трансформер архитектуре на област рачунарског вида

Три године након објављивања поменутог рада, појављује се нови рад чији су аутори такође из компаније *Google* [4], а истражује примену трансформер архитектуре у области рачунарског вида. Овај модел је назван ***Vision Transformer (ViT)*** и представља адаптацију већ постојеће архитектуре на нови домен, односно, обраду слика. ViT значајно проширује границе примене трансформер модела на нове врсте података, пружајући ефикасна и скалабилна решења за задатке везане за препознавање слика на великим размерама.

Настављајући тренд иновација, појављују се и други модели засновани на трансформер архитектури прилагођени рачунарском виду. *DeiT* (*Data-efficient image Transformer*) је један од њих, фокусиран на ефикасно коришћење података и ресурса приликом обраде слика. Овај модел се истиче високом прецизношћу уз смањену потрошњу ресурса, чинећи га посебно корисним у ситуацијама са ограниченим ресурсима.

Уз *DeiT*, појављује се и *Swin Transformer* који уводи хијерархијску организацију пажње путем "*window-based*" механизма. Овај приступ омогућава моделу да ефикасније анализира различите делове слике, што је кључно за суочавање са све већим и сложенијим скуповима података у домену рачунарског вида.

Трансформери се, за разлику од конволуционих неуронских мрежа, не ослањају на обраду секвенцијалних података путем клизног прозора. Уместо тога, користе механизам само-пажње, који им омогућава да симултано узму у обзир целу секвенцу, чинећи сваки независни улаз свестраним у односу на све друге улазе. Ова карактеристика чини трансформере изузетно ефикасним у моделирању далеких зависности у подацима, што је кључни изазов у задацима обраде природних језика, сажимања текста и других. Брзо су нашли примену и у домену рачунарског вида, постепено превазилазећи традиционалне *ConvNet* моделе.

Као што је речено на почетку овог рада, разлог због којег су конволуционе неуронске мреже надмашиле вишеслојне перцептроне (скр. *MLP*) био је тај што смо ограничили број

суседних пиксела слике који могу утицати на пиксел помоћу малих конволуцијских филтера, док су вишеслојне перцептрони, с друге стране, сматрали све пикселе суседима. Трансформери чине овај процес разматрања суседа научивим, користећи *MLP*-ове који практично могу дати мрежи способност разматрања сваког пиксела као суседа, уз истовремено коришћење механизма пажње. То се затим надограђује *MLP*-овима који дају мрежи способност учења који су пиксели важнији, а који мање важни. У овом раду, архитектура визуелних трансформера је централна тема. С тим у вези, у наставку, биће детаљно разматрана ова архитектура, укључујући све њене индивидуалне компоненте и начин рада.

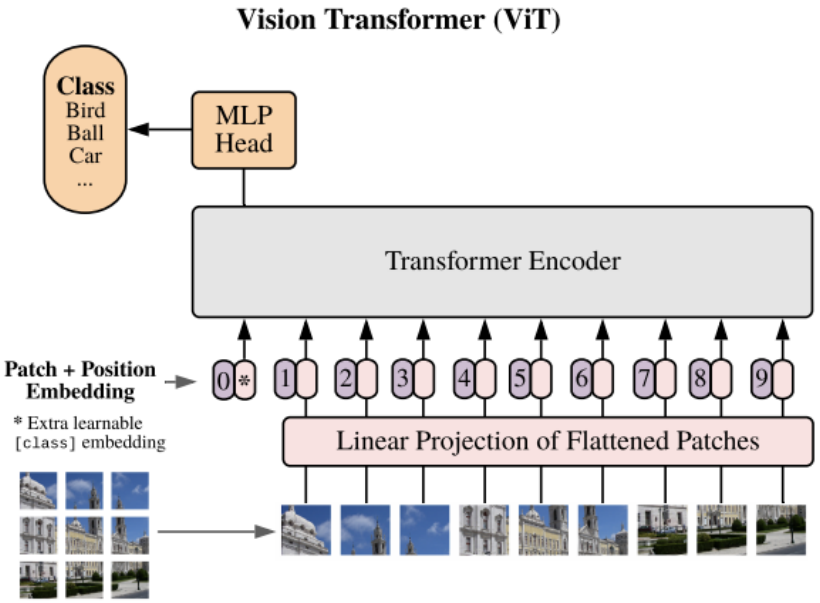
3 Визуелни трансформери

Ово поглавље посвећено је визуелним трансформерима, иновативним моделима заснованим на трансформер архитектури, који су прилагођени за обраду слика. Фокус ће бити на детаљној анализи кључних компоненти и принципа *ViT* архитектуре како бисмо дубље разумели њихов значајан допринос у области рачунарског вида. Као што је раније споменуто, овај модел се први пут појавио у раду [4] и као такав представља први модел овог типа у свету рачунарског вида. Поред овог модела, биће анализиране и његове алтернативе које су настале у циљу надограђивања постојећег модела.

У наредним секцијама, детаљно ћемо анализирати сваки блок *ViT* архитектуре, почевши од подела на печеве, њиховог поравнавања и трансформације путем потпуно повезаних слојева (енг. *Fully Connected Layers*). Затим, проучићемо улогу *position embedding*-а и класног токена. Посебан нагласак биће стављен на трансформер енкодер, где ћемо детаљно истражити *Multi-Head-Attention* блок, *MLP*, нормализацију и резидуалне везе. На крају, анализираћемо улогу *MLP head*-а у финалном процесирању резултата. Кроз ову детаљну анализу, читаоци ће стећи дубоко разумевање сваке компоненте *ViT* архитектуре и њиховог међусобног доприноса у ефикасном препознавању и анализи слика.

3.1 *ViT* архитектура

ViT (*Vision Transformer*) представља јединствени пример трансформер модела у контексту обраде слика, будући да се састоји искључиво од структуре енкодера, што га разликује од традиционалних архитектура које укључују и декодере. Ова специфичност наглашава снажан фокус *ViT* модела на ефикасној репрезентацији и анализи визуелних информација путем трансформер енкодера. На слици испод, приказана је архитектура овог модела, слика је преузета из оригиналног рада који је и представио ове моделе [4].



Слика 3.1.1 *Vision Transformer* архитектура [4]

Процес обраде слике започиње сегментацијом исте на квадратне печеве (енг. *Patch*), које потом трансформишемо у једнодимензионални низ. Субсеквенцијално се примењује линеарна пројекција на сваки печ ради пресликавања у одговарајућу димензионалност.

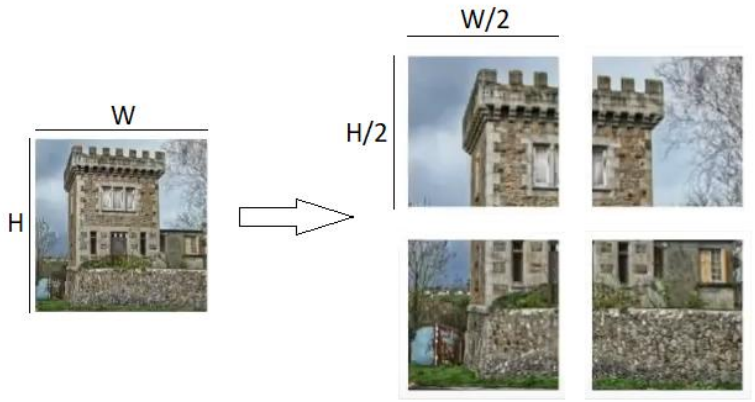
Међутим, разбијање слике у једнодимензионални низ неизбежно поставља изазов у разумевању односа између појединачних печева и њихове локације на слици. У циљу превазилажења овог проблема, имплементира се механизам у којем се додаје додатна информација сваком од печева, познат под називом „уграђивање позиције“ (енг. *position-embedding*), које је, на слици, репрезентовано љубичастим нумеричким обележјима од 1 до 9. Додатни детаљ архитектуре укључује примену технике инспирисане обрадом природног језика, где се уводи додатна учљива информација, често реферисано као токен класе (енг. *Class token*), који се сматра додатним улазом при чему му се додељује позиција 0 у секвенци. [5]

Након ових прелиминарних корака, низ уграђивања (енг. *embeddings*) се прослеђује трансформер енкодеру како би спровео своју трансформациону операцију. На последњем кораку, издваја се *embedding* са позиције 0 и прослеђује кроз сложени вишеслојни перцептрон (*MLP*) који се користи за класификацију улазне слике. Сви ови кораци ће бити темељно описани у наставку.

3.1.1 Модул за креирање и трансформацију печева

С обзиром на то да поменути трансформер модел обрађује искључиво податке у секвенцијалном формату, ово представља специфичан изазов у области рачунарског вида. Разлог овоме је тај што су, у овом случају, подаци најчешће троканалне (*RGB)* слике. Сходно овоме, неопходно је пронаћи начин конверзије ове слике у секвенцу (низ) токена. Ово задужење има модул у трансформер архитектури који се назива модул за креирање и трансформацију печева (енг. *Patch Embedding Module*). Овај модул трансформише улазну слику у секвенцу печева.

Претпоставимо да је ширина слике *W*, а висина *H*. Иницијално, слику делимо на печеве висине *H*/2 и ширине *W*/2. Димензије печева су узете за потребе илустрације. У реалности, величина печа варира, при чему је у оригиналном раду коришћен печ величине 16x16 пиксела. Ова величина печа диктира и сам број печева који ће бити коришћен у даљој обради. У овом случају, број печева је 4. На слици испод се може видети како то заправо изгледа.



Слика 3.1.1.1 Подела слике на печеве

Сада, уместо да третирамо ове печеве као неку врсту 2*D* мреже, користићемо их као низ формиран узимањем печева по редоследу дуж колона. Ово се може видети на слици 3.1.1.2. На овај начин, у нашем случају, добијамо секвенцијални излаз величине 4.

Слика 3.1.1.2 Секвенца печева

С обзиром на то да је сваки печ у овом тренутку 2*D* матрица која је састављена од 3 канала (*R, G* и *B* канал), добијамо да је укупна величина сваког печа 3 x *H*/2 x *W*/2, или, у општем случају: *Број_канала x Висина_печа x Ширина_печа*. Трансформер архитектура овај проблем решава једноставним поравнавањем наведеног троканалног печа (матрице) у један вектор. На овај начин, добијени вектор садржаће прво вредности *R, G* и *B* канала за први пиксел, потом за други итд. Применом овог поступка над свим печевима, добићемо заједнички улаз величине представљене у формули 3.1.1.1, при чему *N* представља број печева:

$$N \text{ x } (Број\_канала * Висина\_печа * Ширина\_печа). \qquad (3.1.1.1)$$

У наредном кораку, неопходно је извршити усклађивање димензија са захтевима трансформера. Наиме, трансофрмер архитектура обрађује податке који имају унапред дефинисану димензију. Означићемо ту димензију са *D*. Због овога, потребно је наш улаз свести на облик *N* x *D*. У нашем случају важи, *N*=4.

Како бисмо ово постигли, уводимо потпуно повезани слој (енг. *Fully Connected Layer*) који ће имати *D* неурона и матрицу тежина W_t величине (*Број_канала*Висина_печа*Ширина_печа*) x *D*. На овај начин, на излазу добијамо матрицу величине, *N* x *D*, где свака врста представља векторску репрезентацију одговарајућег печа,

тј. такозваног печ уграђивања (енг. *patch embedding)*. У даљем тексту ће се, лакшег разумевања ради, користити одговарајући енглески назив.

Важно је напоменути да тренутно добијене репрезентације печева не садрже никакве информације о свом положају, те је потребно некако убацити информације о позицији у ове репрезентације. Ово је неопходно урадити, с обзиром на то да трансформер не обрађује секвенцу печева једну по једну, већ истовремено у паралели.

3.1.2 Модул за додавање позиционе информације

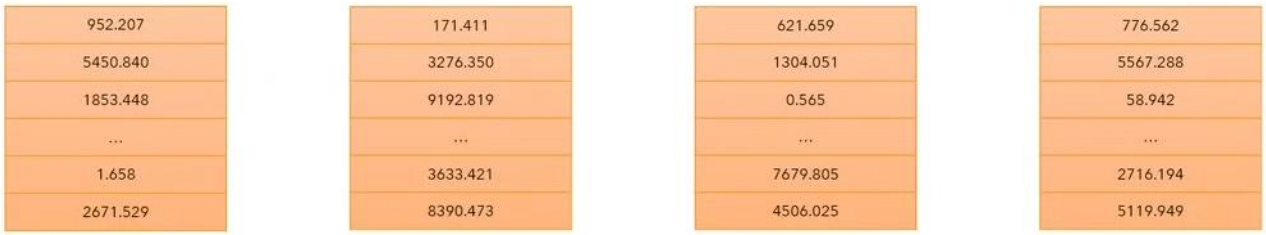
Као што је већ напоменуто, потребно је додатно обогатити сваки тренутни *patch embedding* вектор додатном информацијом о локалитету, односно о позицији унтар почетне улазне слике. Ово се дешава непосредно пре прослеђивања печева енкодеру трансформер модела.

Овај приступ произилази из потребе да свака операција унутар трансформера посматра улаз као скуп (сет) података, без узимања у обзир позиције и редоследа. На пример, уколико бисмо променили редослед улазних печева, то би аутоматски имало утицаја и на излаз. Ово, наравно, представља изазов јер смо, разбијањем улазне слике на печеве, изгубили информацији о њиховим релативним позицијама.

Решење лежи у коришћењу позиционих уграђивања (енг. *position embeddings*) која би на својеврстан начин означила сваки печ сопственом позицијом. Као и код *patch embedding* вектора, и у овом случају ћемо надаље користити енглески назив. На овај начин трансформер би имао информацију о томе, условно речено, где се налази сваки печ и шта тачно „посматра“. Наравно, поставља се питање, на који начин извршити ово означавање на начин да то трансформер (тачније енкодер) разуме.

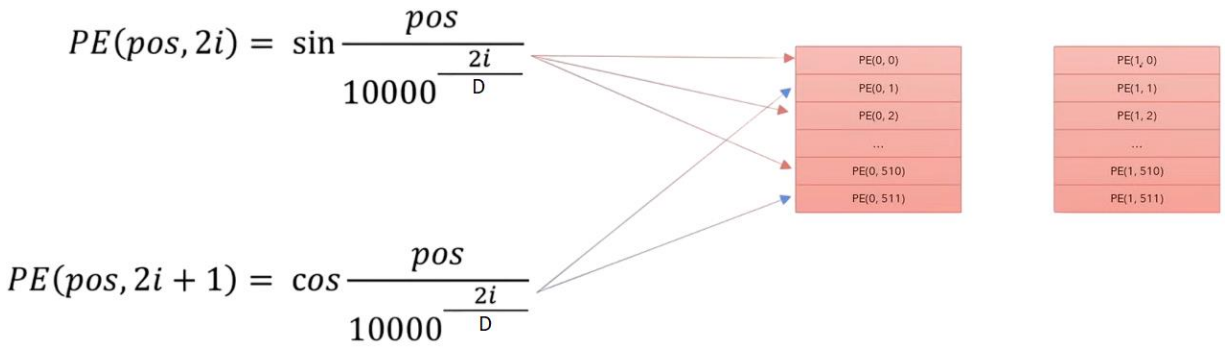
Наиме, у раду [4] истраживачи су покушали додавање позиционих информација на различите начине. Један од приступа био је да се једноставно дода број индекса врсте матрице (матрица чије врсте сачињавају *patch embedding* вектори одговарајућих печева). Међутим, наводи се у даљем тексту, да се ова метода није довољно добро показала, те су се окренули другим решењима.

Битно је напоменути да постоји много приступа и начина на који се овај проблем може решити. Уместо претходно поменутог примера, одлучено је да се користе вектори. Ово је, вероватно, најбоље објаснити на примеру. На слици испод су приказана хипотетички *patch embedding* вектори. Претпоставимо да се ови вектори односе на печеве из претходног потпоглавља приказаних на слици 3.1.1.2.



Слика 3.1.2.1 *patch embedding* вектор сваког од печева, величине *D*

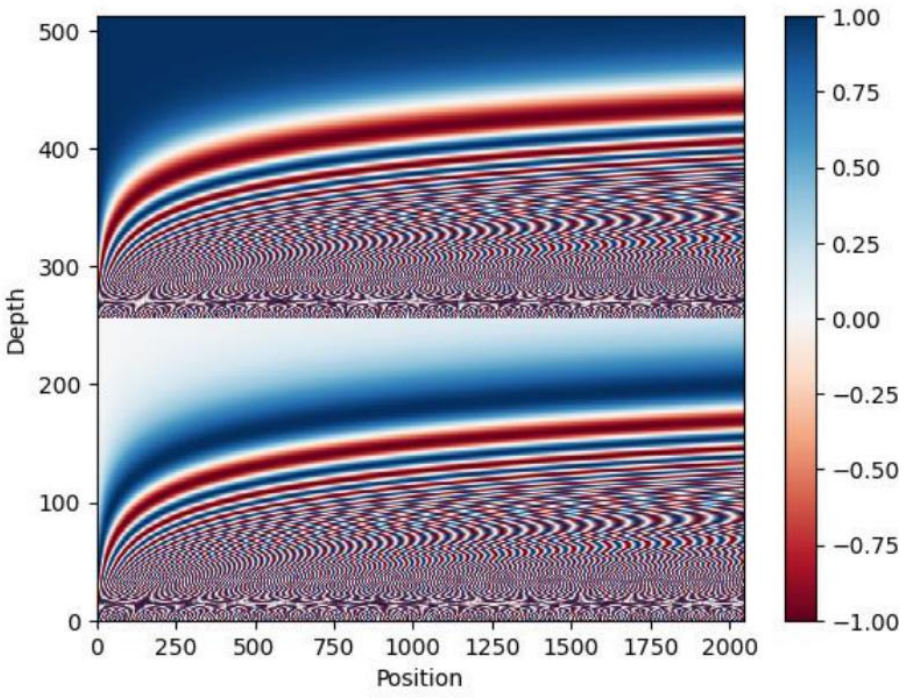
Оно што је потребно урадити јесте креирати нове векторе (*position embedding* векторе), чије ће величине бити *D*. За сваку позицију унутар ових вектора неопходно је одредити вредност по формули наведеној на слици испод. Ова формула се, између осталог користи и у раду који је споменут раније у тексту.



Слика 3.1.2.2 Позициона уграђивања

Ознаке на овој слици су следеће: *pos* у овој формули означава позицију печа у секвенци печева, док са друге стране 2*i* представља позицију елемента унутар самог вектора. За парне позиције рачуна се синусна, а за непарне позиције косинусна формула. Веома је битно напоменути да се добијени вектор рачуна само једном, односно, не долази до поновног израчунавања за сваку улазну слику. С обзиром на то да се слике деле на фиксни број печева, сваки печ ће имати унапред дефинисани *position embedding* вектор, који ће бити исти за сваки одговарајући печ. Ово, наравно, представља велики корак у погледу побољшања ефикасности трансформер архитектуре.

Зашто баш тригонометријске функције? Тригонометријске функције попут косинуса (*cos*) и синуса (*sin*) природно поседују образац који модел може препознати као континуиран, па су релативне позиције лакше уочљиве за модел. Обе функције су периодичне и сходно томе могу да представе позицију за било коју величину секвенце на конзистентан начин. Такође, ограничене су вредностима између -1 и +1 што је корисно како би се избегле велике вредности које би могле имати лош утицај на механизам самопажње. Посматрањем графика ових функција (слика 3.1.2.3), способни смо и ми (људи) приметити образац, па је емпиријски закључак да ће га и сам модел препознати.



Слика 3.1.2.3 Илустрација тригонометријских функција

Када се успешно креирају *position embedding* вектори, следи сабирање са *patch embedding* векторима (векторима који су исте величине као и новокреирани вектори). На овај начин се у досадашње *patch embedding* векторе усађује и позициони контекст. Шема је приказана на следећој слици.

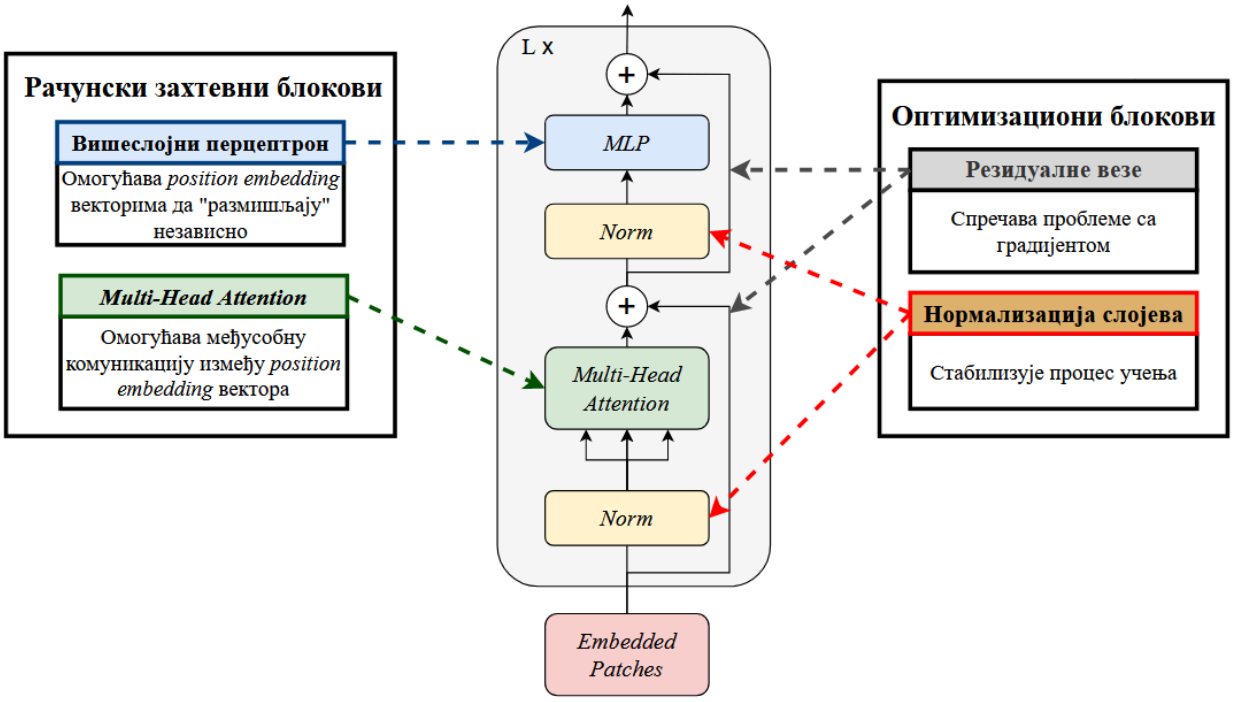
Слика 3.1.2.3 Добијање улаза у енкодер

Након што добијемо ове векторе, трансформер енкодер ће моћи да схвати локалне карактеристике сваког улазног печа. Ови унапређени вектори, сада обогаћени релевантним локалним и просторним информацијама, затим се преносе следећем сегменту у низу трансформер архитектуре, односно трансформер енкодеру. Важно је напоменути да у овом кораку такође уводимо класни токен, посебан токен преузет из области обраде природних језика, који се користи за коначну класификацију. Класном токену се додељује позиција 0, чиме се обезбеђује посебно разматрање у процесу трансформације слике. Овај додатак има кључну улогу у омогућавању моделу да придружи одговарајућу класу финалном излазу.

Узимајући у обзир да се истраживање у домену позиционих уграђивања непрестано развија, пружа се континуирана прилика за даље унапређење и прилагођавање у специфичним контекстима примене. Ово одржава динамичност поља рачунарског вида и указује на потребу за иновацијама како би се ефикасније интегрисале просторне информације унутар модела. У даљем тексту ћемо се усредсредити на детаљну анализу блоковске структуре трансформер енкодера.

3.1.3 Трансформер енкодер

Трансформер енкодер започиње свој процес од *position embedding* вектора. Енкодер је конципиран као низ блокова, постављених један изнад другог. Сваки блок је идентичан и састављен је од низа мањих подблокова. У приказу дијаграма на слици испод, можемо да видимо споменуте подблокове. У циљу темељног разумевања овог поступка, анализираћемо сваки од ових подблокова посебно.



Слика 3.1.3.1 Подблокови трансформер енкодера

На слици видимо да постоје 4 кључна подблока. Два подблока су кључна у погледу израчунавања (енг. *Compute Heavy Blocks*), остала два спадају под ткзв. оптимизационе подблокове. Као што је и на слици назначено, у прву категорију спадају *Multi-Head Attention* и вишеслојни перцептрон, док се друга категорија односи на нормализацију слојева (означена са *Norm* на слици) и резидуалн везе.

Апстрактно гледано, блок са више глава пажње омогућава међусобну комуникацију и размену информација између различитих *positon embedding* вектора, док вишеслојни перцептрон омогућава сваком вектору независно размишљање о информацијама које је апсорбовао од својих суседа. Ови блокови су изузетно рачунски захтевни и представљају главно место где се усмерава готово сва рачунска снага.

Поред ових, укључена су и додатна два блока како би се олакшао процес оптимизације. Први блок је имплементиран у облику резидуалних веза, док други представља слој нормализације. Оба блока појављују се по два пута. Резидуалне везе доприносе слободном протоку градијената, док нормализација слоја има за циљ стабилизацију процеса учења. У наредним поглављима следи детаљан опис функционалности и начина рада сваког од назначених потблокова.

3.1.3.1 *Single-Head Attention* механизам

Multi-Head Attention блок представља кључну компоненту трансформер енкодера и као таква има веома важну улогу у целој трансформер архитектури уопште. Ипак, због његове инхерентне комплексности, разумевања ради, најпре је неопходно детаљније размотрити концепт само-пажње (енг. *Self-attention*). Важно је напоменути да је овај концепт постојао много пре појаве трансформера, али га је трансформер архитектура значајно унапредила и доделила му нове домене примене. Са развојем ове архитектуре добио је и нову терминологију, те је у овом контексту познатији под именом *Single-Head Attention*. [4]

Механизам само-пажње представља концепт који омогућава печевима да међусобно комуницирају. Дакле, као што је раније споменуто, на самом улазу у трансформер енкодер налазе се *position-embedding* вектори (матрица чије су врсте поменути вектори). Ови вектори тренутно не садрже никакве информације о другим векторима. Контекст којим тренутни вектори располажу се завршава са познавањем својих вредности и позиције у оквиру улазне слике. Међутим, с обзиром на то да се сва израчунавања унутар трансформер енкодера врше над свим печевима истовремено, потребно омогућити векторима да продубе свој контекст у погледу познавања своје релевантности (сличности) са другим векторима.

Механизам само-пажње креће креирањем 3 нова вектора за сваки улазни *position-embedding* вектор, тј, једноставније гледано, креирањем 3 нове матрице чије су врсте одговарајући вектори. Иницијално, ови вектори су потпуно идентични као и одговарајући улазни вектори. Називи ових вектора су упити (енг. *Queries*), кључеви (енг. *Keys*) и вредности (енг. *Values*). [3]

Након иницијализације, следи пројекција ових вектора. Пројекција се врши коришћењем линеарних слојева неуронских мрежа чији су називи *Query Network, Key Network* и *Value Network*, респективно. Ове слојеве неуронске мреже можемо представити њиховим матрицама тежина:

1) $W^Q \in \mathbb{R}^{D \times d_k}$ (Матрица пројекције на *Queries* вектор),
2) $W^K \in \mathbb{R}^{D \times d_k}$ (Матрица пројекције на *Keys* вектор) и
3) $W^V \in \mathbb{R}^{D \times d_v}$ (Матрица пројекције на *Values* вектор).

У наведеним матрицама, *D* представља димензију коју трансформер прихвата и са којом извршава све своје операције, *dk* димензију *Queries* и *Keys* вектора, а *dv* димензију *Values* вектора. Сходно овоме, одговарајуће векторе (матрицу) добијамо једноставним матричним множењем, тј:

1) $Q = XW^Q \in \mathbb{R}^{N \times d_k}$, (3.1.3.1.1)
2) $K = XW^K \in \mathbb{R}^{N \times d_k}$ и (3.1.3.1.2)
3) $V = XW^V \in \mathbb{R}^{N \times d_v}$ (3.1.3.1.3)

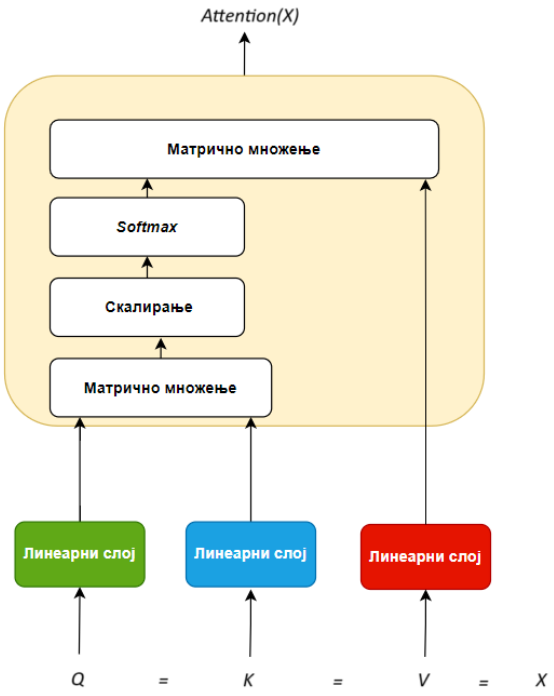
Q, K и *V* представљају новодобијене матрице, чије су врсте састављене од индивидуалних *Queries, Keys* и *Values* вектора, респективно. Ови вектори одговарају пројектованим верзијама одговарајућих врста у матрици *X*. Матрица *X* је матрица улазних *position-embedding* вектора. Као што се може приметити, димензије излазних матрица у случају *Q* и *K* су (N x d_k), док је димензија матрице *V*, (N x d_v). Важно је напоменути да се, у пракси, димензије d_k и d_v скоро увек поклапају, те важи $d_k = d_v$.

Сада, када смо израчунали потребне матрице, можемо прећи на следећи корак, тј. корак израчунавања само-пажње (*scaled dot-product attention* [3]). Ово рачунамо коришћењем следеће формуле:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \in \mathbb{R}^{N \times d_v} \quad (3.1.3.1.4)$$

Разумевања ради, вероватно је једноставније објаснити векторе (*Queries, Keys* и *Values*) на нивоу интуиције. Због тога, употребићемо једноставно поређење. Рецимо да желимо да претражимо нешто на интернету. Оно што куцамо можемо поистоветити са *Query* вектором. Резултате који ће се појавити (наслови предложених страница) можемо тумачити као *Key*, а изабрану страницу и њен садржај можемо схватити као *Value*.

Слично овоме, индивидуални печ, пре уласка у трансформер енкодер, нема информацију о другим печевима. Међутим, да би имао увида у то шта представља и са којим печевима има највише повезаности, он тражи, путем свог *Query* вектора, одговарајућа поклапања са свим другим *Key* векторима, на тај начин добијајући сличност између њих. Формула 3.1.3.4, као и цео поступак који јој претходи (креирање *Q, K* и *V* матрица коришћењем линеарних слојева) илустровани су на слици испод.



Слика 3.1.3.1.1 Single-Head Self Attention

Унутар заграде *softmax* функције (део формуле пре матричног множења са матрицом *V*) користи се специфичан математички израз због уске повезаности с косинусном сличношћу. Разлог за коришћење овог израза лежи у начину на који се косинусна сличност између два вектора израчунава. [6]

$$sim(A,B) = \cos\theta = \frac{A \cdot B}{\|A\|\|B\|} \qquad (3.1.3.1.5)$$

При чему је θ угао између вектора A и B. Ова формула се може написати и на следећи начин:

$$sim(A,B) = \frac{A \cdot B}{scaling} \qquad (3.1.3.1.6)$$

Међутим, с обзиром на то да се у нашем случају јављају матрице уместо вектора, потребно је транспоновати матрицу B како би се задовољили сви услови матричног множења. Сада, када заменимо векторе A и B адекватним матрицама, добијамо следећу, коначну формулу:

$$sim(Q,K) = \frac{Q \cdot K^T}{scaling} \qquad (3.1.3.1.7)$$

Када бисмо *scaling* заменили са $\sqrt{d_k}$, добили бисмо полазну формулу која се налази унутар *softmax* функције. Разлог коришћења управо ове вредности за *scaling* је тај што је *softmax* функција веома сензитивна тј. не даје корисне градијенте када је један улаз много

већи од других. Због овога, потребно је поделити са $\sqrt{d_k}$ како би се избегле високе магнитуде (енг. *"peaky" affinites*) када су унутрашњи производи велики. На овај начин, варијансу одржавамо на вредности око јединице.

На крају, у циљу давања комплетне слике рада механизма само-пажње, приказан је илустративни пример. Прегледности ради, наставићемо са започетим примером наведеним у поглављу 3.1.2.

	position embedding 1	position embedding 2	position embedding 3	position embedding 4
position embedding 1	0.268	0.119	0.134	0.148
position embedding 2	0.124	0.278	0.201	0.115
position embedding 3	0.147	0.132	0.262	0.097
position embedding 4	0.210	0.128	0.206	0.212

Слика 3.1.3.1.2 Први део формуле

Дакле, имамо 4 улазна печа и трансформер који ради са векторима димензије 512. Сходно овоме, матрице *Q* и *K* ће бити величине (4, 512). Након примене првог дела формуле 3.1.3.4 (део који претходни коначном множењу са матрицом *V*) добијамо матрицу која је величине (4, 4). Добијена матрица нам указује на сличност између сваког од *position-embedding* вектора. Очекивано, на главној дијагонали ове матрице се налазе највеће вредности јер је, у овом случају, косинусна сличност израчуната између истих вектора. Битно је напоменути да је сума појединачних свих редова једнак јединици. Ово је директна последица *softmax* функције која рачуна вероватноће у зависности од улазних вредности.

Следи корак множења тако добијене матрице са *V* матрицом. На овај начин добијамо такозвану матрицу пажње (енг. *attention matrix*). Ова матрица има исте димензије као и матрице пре почетка израчунавања, тј. (4, 512). Овог пута, врсте ове матрице се такође односе на *embedding* векторе на које су се односиле и раније, с тим што су вектори сада проширили свој контекст. Сада обједињују претходни контекст који се тицао позиције печева на улазној слици и контекст који говори о значају других вектора на посматрани вектор. Приказ овог дела се може видети на слици испод.

	position embedding 1	position embedding 2	position embedding 3	position embedding 4
position embedding 1	0.268	0.119	0.134	0.148
position embedding 2	0.124	0.278	0.201	0.115
position embedding 3	0.147	0.132	0.262	0.097
position embedding 4	0.210	0.128	0.206	0.212

X

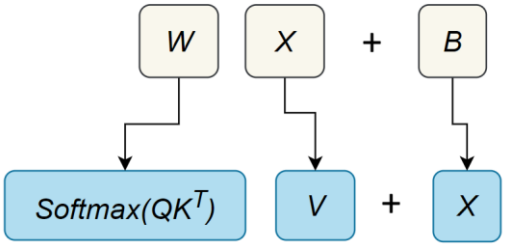
V
(4, 512)

=

Attention
(4, 512)

Слика 3.1.3.1.3 Други део формуле

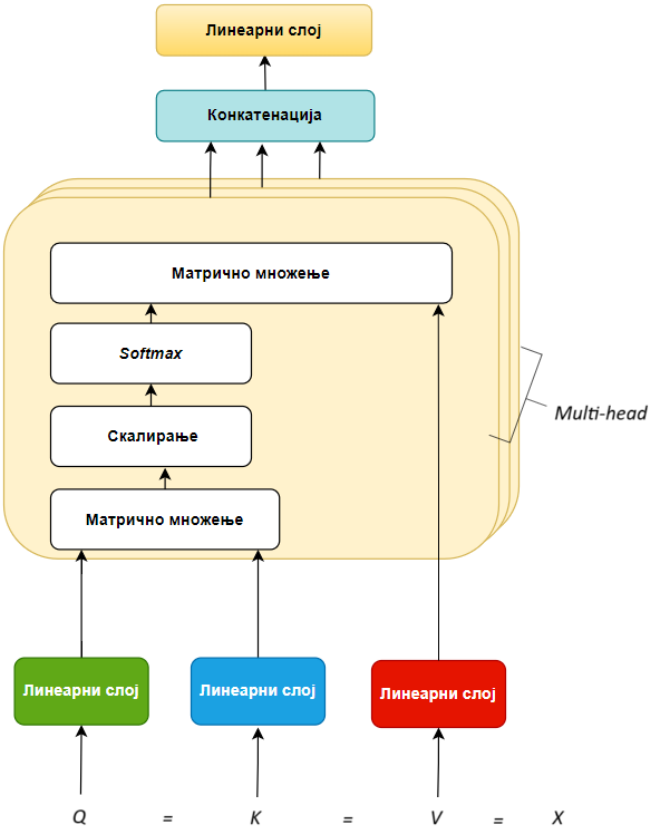
Ако извршимо поређење слоја само-пажње са обичним линеарним слојем, видећемо колико је заправо овај механизам моћан. Наиме, општа формула за израчунавање излаза из једног линеарног слоја је $WX + B$. У случају механизма само-пажње формулу можемо да поистоветимо са наведеном формулом, с тим што би се у овом случају W рачунало за сваки улаз посебно (W је директно зависно од улаза) за разлику од формуле линеарног слоја где имамо фиксну матрицу. Такође, и B директно зависи од улаза. Ово значи да уз помоћ само-пажње ми, практично, тренирамо адаптивни слој који манипулише сопственим тежинама зависно од улазних података. На слици 3.1.3.1.4 је илустровано ово поређење.



Слика 3.1.3.1.4 Поређење само-пажње и линеарног слоја

3.1.3.2 *Multi-Head Attention* блок

Сада, када је појашњен начин функционисања *Single-Head Attention* механизма, коначно се може размотрити *Multi-Head Attetnion* блок, који је саставни део трансформер енкодер архитектуре. Начин рада овог блока, као што се и очекује, има велики степен повезаности са иницијалним *Single-Head Attention* механизмом. Ова сличност се може уочити на слици испод.



Слика 3.1.3.2.1 Multi-Head Attention процедура

Како би се ова процедура реализовала, користи се *H* одвојених глава пажњи (енг. *attention heads*). За сваку од ових глава се користи одвојена матрица за креирање *Queries*, одвојена матрица за креирање *Keys* и одвојена матрица за креирање *Values*. Као и раније, потребно је применити формулу за израчунавање матрице пажње ради креирања излаза. Када се овај процес заврши за свих *H* глава, врши се конкатенација добијених резултата након које се врши финална пројекција. Финална пројекција се користи ради свођења резултата на одговарајуће димензионалности. Овај процес се може видети у псеудокоду испод.

Алгоритам 1 *Multi-Head Attention*

Улаз: H, $X, {W_h}^Q, {W_h}^K, {W_h}^V, d_k$

Излаз: Матрица пажње

1: **for** $h = 1, ...H$ **do**
2: $\quad Q_h = X{W_h}^Q$
3: $\quad K_h = X{W_h}^K$
4: $\quad V_h = X{W_h}^V$
5: $\quad \text{head}_h = \text{softmax}\left(\frac{Q_h K_h^T}{\sqrt{d_k}}\right) V_h$
6: **end for**
7: $MultiHead(X) = Concat(head_1, ..., head_h)W^O$

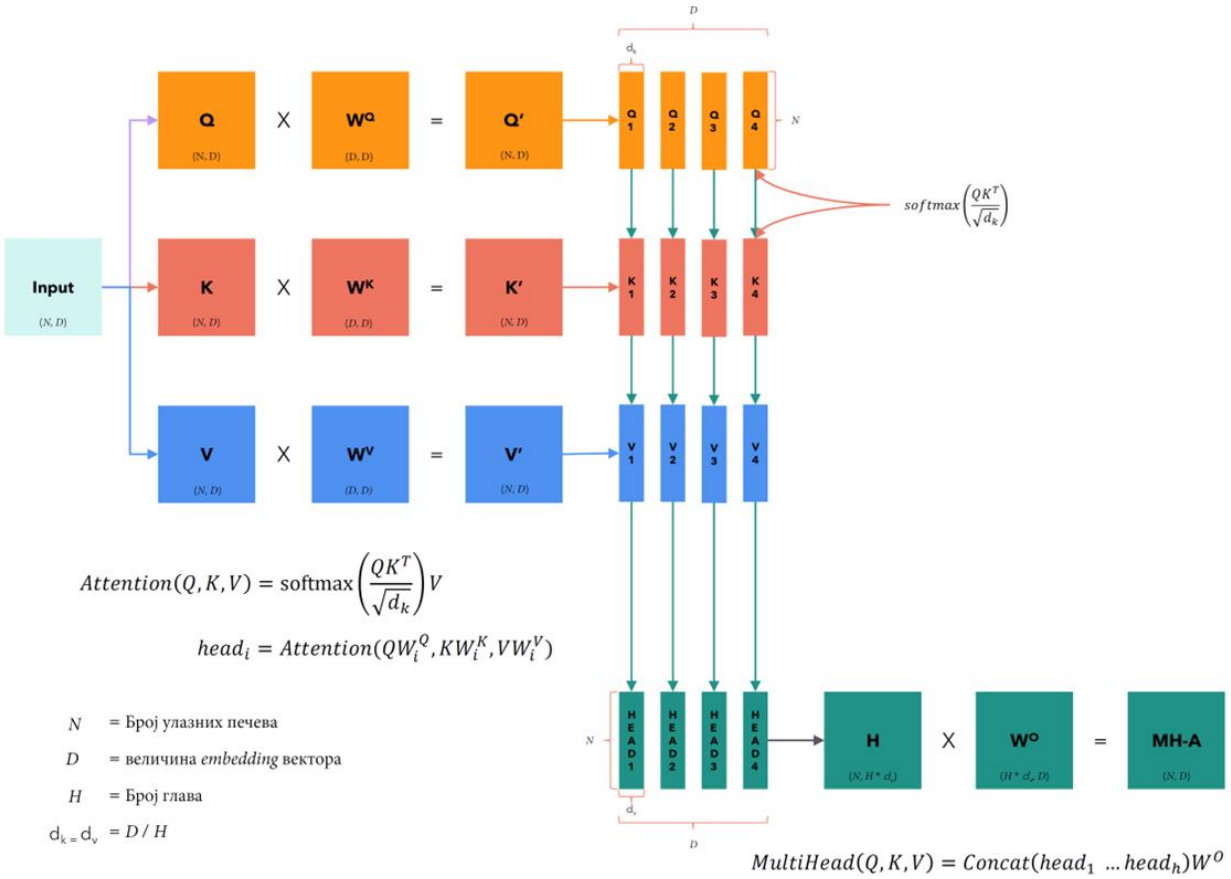
Псеудокод 3.1.3.2.1 Multi-Head Attention

Иако се на први поглед чини, *for* петља заправо није спора и компликована јер се, у пракси, свака глава пажње извршава паралелно. У пракси се, такође, за димензију главе најчешће бира мањи број. Најчешће се користи број који је једнак количнику *D* и *H*, тј.

$$d_k = d_v = \frac{D}{H} \qquad (3.1.3.2.1)$$

На крају, потребно је дотаћи се и асимптотске комплексности овог алгоритма. Ако занемаримо пројекције, асимптотска сложеност је $O(N^2 * D)$. Ово указује на то да сложеност овог алгоритма расте у односу на дужину секвенце (D). Ово је сложеност која није занемарљива, те су стога многи истраживачи предлагали стратегије за смањење ове комплексности. Касније ће бити споменути трансформер модели чија архитектура и начин рада потенцијално могу смањити ову сложеност.

Ово потпоглавље ће бити завршено са једним илустративним примером у циљу темељног разумевања овог блока трансформер енкодер архитектуре.



Слика 3.1.3.2.2 Multi-Head Attention процедура

Величина *embedding* матрице за сваки печ унутар архитектуре визуелног трансформера је величине (4, 512). Током ове дискусије, појам "*D*" се користи да означи величину *embedding* вектора, која је, у овом случају, постављена на 512. Погледајте слику 3.1.3.2.2 за визуални приказ, где су ознаке експлицитно приказане у легенди која се налази у доњем левом углу пратеће слике.

Као што је илустровано на слици 3.1.3.2.2, улаз пролази кроз одређену трансформацију. То укључује репликацију улаза четири пута, при чему једна копија следи прескачућу везу (више о томе касније), а преостале три су усмерене ка различитим гранама механизма пажње са више глава. Свака грана добија јединствено име које означава њену улогу у наредним корацима процесирања. Увођење резултирајућих матрица, названих *Q*, *K* *V*, је део процеса.

Механизам пажње са више глава функционише множењем матрица *Q*, *K* и *V* са матрицама параметара означеним као W^Q, W^K и W^B, респективно. Ове матрице параметара имају димензије (*D*, *D*), усклађујући се са величином *embedding* матрице. Резултат овог множења даје матрице назване *Q'*, *K'* и *V'*, задржавајући димензије (*N*, *D*).

Настављајући даље, слика 3.1.3.2.2 помаже у визуализацији наредних корака, наглашавајући раздвајање матрица дуж димензије *D*. Свака глава унутар механизма пажње са више глава процесуира целу слику фокусирајући се на одређени подскуп *embedding* вектора за сваки печ.

На слици 3.1.3.2.2 дат је и јасан приказ израчунавања пажње између мањих матрица, означених као *HEAD* 1 до *HEAD* 4. Димензије ових глава су секвенциране по d_v, при чему је d_v еквивалентно d_k (*D* подељено бројем глава, *H*). Субсеквенцирање ових глава укључује њихово спајање дуж димензије d_v, што на крају резултира матрицом димензија (N, $H*d_v$).

Слика 3.1.3.2.2 додатно појашњава последњу фазу процеса механизма пажње са више глава, укључујући множење конкатенираног резултата матрицом W^O. Димензије ове матрице су ($X*d_v$, D), на крају дајући нову матрицу која представља исход механизма пажње са више глава, са димензијама (N, D).

Ова секција пружа свеобухватан увид у механизам пажње са више глава. Јединствени приступ раздвајања матрица дуж димензије *D* омогућава свакој глави да се фокусира на различите аспекте истог сегмента слике, подстичући нијансирану анализу визуелних карактеристика, аналогно језичким нијансама ухваћеним од стране језичких глава у трансформаторима.

3.1.3.3 Блок вишеслојног перцептрона

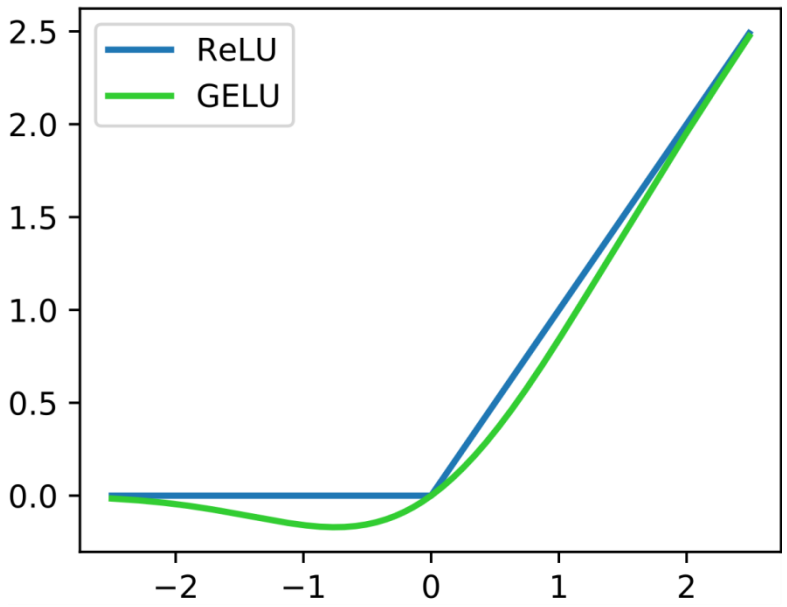
Проучавајући сложености блока енкодера архитектуре овог визуелног трансформера, темељно је приказана позадина *Multi-Head Attention* блока, компоненте широко признате као најкомплексније унутар архитектуре. У овом потпоглављу биће више речи о вишеслојном перцептрону, следећем рачунски интензивном блоку у овој архитектури. Интуитивно гледано, након размене сазнања између *position embedding* вектора, која се десила у претходно описаном блоку, поставља се императив око њиховог независног „размишљања“ о ономе што су до сада научили. Иза ове интуиције стоји управо блок вишеслојног перцептрона.

Блок вишеслојног перцептрона је, у контексту архитектуре визуелних трансформера, организована укључивањем двослојног перцептрона. Овај двослојни перцептрон је реализован коришћењем два линеарна слоја са додавањем нелинарности првом линеарном слоју. У наставку је дата формула овог двослојног перцептрона.

$$MLP(x) = W_2\sigma(W_1x + b_1) + b_2 \qquad (3.1.3.3.1)$$

У овој формули, *x* представља улазни вектор, W_1 и b_1 су тежине (матрица тежина) и *bias* првог слоја неуронске мреже, σ представља активациону функцију (увођење нелинеарности у први слој), W_2 и b_2 су тежине и *bias* другог слоја неуронске мреже. Овај израз описује начин на који улазни подаци (вектор) пролазе кроз овај двослојни перцептрон.

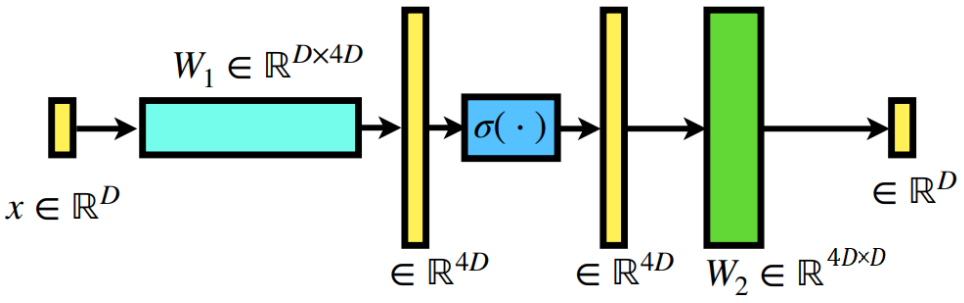
Оригинална трансформер архитектура користила је *ReLU* (*Rectified Linear Unit)* као своју активациону функцију. Са друге стране, архитектура визуелних трансформера се, у пракси, најчешће ослања на коришћење *GeLU* (*Gaussian Error Linear Unit*) активационе функције. Слика 3.1.3.3.1 приказује ове две активационе функције.



Слика 3.1.3.3.1 *ReLU* и *GeLU* активационе функције

Као што се може видети на овом графику, *GeLU* има неколико значајних разлика у поређењу са *GeLU*. Ова неконвексна, немонотона функција није линеарна у позитивном домену и показује закривљеност у свим тачкама. С друге стране, *ReLU*, која је конвексна и монотона активациона функција, линеарна је у позитивном домену. Знајући ово, повећана закривљеност и немонотоност могу омогућити *GeLU* да лакше апроксимира сложене функције у поређењу са *ReLU*, што је и разлог коришћења у овој архитектури. [7]

У практичној примени овог двослојног перцептрона, уобичајена пракса је коришћење фактора проширења чија је вредност најчешће четири. Ова, емпиријски одређена вредност, показује се веома ефикасном у процесу трансформације. На слици испод је илустрован начин рада ове двослојне неуронске мреже са фактором проширења четири.



Слика 3.1.3.3.2 Илустрација рада двослојног перцептрона

Као што се може видети на слици 3.1.3.3.2, поступак почиње множењем улазног *embedding* вектора величине D са матрицом, означеном са W_1, величине $D\times 4D$. На овај начин се постиже примена горепоменутог проширења и четвороструко повећавање димензије улазног вектора. Након овога, примењује се активациона функција (најчешће *GeLU*), одржавајући исту димензију. Коначни корак обухвата множење другом матрицом, означеном са W_2, на тај начин враћајући димензионалност на почетну (D). Ова стратегија се, до сада, показала као најбоља и самим тим дала најбоље резултате.

Након последњег блока трансформер енкодера, следи један блок који се назива глава вишеслојног перцептрона (енг. *MLP Head)* која се може видети на слици 3.1.1. Овај блок је састављен од једног или више потпуно повезаних слојева са нелинеарним активационим функцијама чији је циљ успешно извршавање класификације слика. Улаз у овај блок је само *embedding* вектор који одговара улазном класном токену. Овај токен је, пролазећи кроз све претходне слојеве, заједно са свим другим векторима, покупио све потребне информације за финалну класификацију.

Овим је детаљно објашњена улога и начин рада свих подблокова који припадају групи рачунски захтевних подблокова. У наредном потпоглављу биће нешто више речи о раније споменутим оптимизационим блоковима трансформер енкодера.

3.1.3.4 Резидуалне везе и нормализација

Након сваког *Multi-Head Attention* блока и блока вишеслојног перцептрона, или, једноставније речено, након сваког подблока у енкодеру, јавља се резидуална веза која је увек праћена блоком нормализације слоја. Тема овог потпоглавља је управо ова врста оптимизације.

Прва резидуална веза на слици 3.1.3.1 (гледано одоздо на доле) одговара једноставном сабирању улаза и резултата који је добијен извршавањем нормализације и *Multi-Head Attention* блока. Са друге стране, друга резидуална веза на истој слици, одговара сабирању претходног излаза и коначног излаза. Следе математички формуле споменутих резидуалних веза, респективно.

$$Y = X + MHA(Norm(X)) \qquad (3.1.3.4.1)$$

$$Out = Y + MLP(Norm(Y)) \quad (3.1.3.4.2)$$

Разлог због којег овај механизам потпомаже процесу оптимизације трансформер енкодера је емпиријске природе. Између осталог, оригинални аутори ових веза наводе следећу реченицу у њиховом раду: „*Претпостављамо да је лакше оптимизовати резидуално мапирање него оригинално, нереференцирано мапирање*“.[5] Свакако јесте тачно да, емпиријски, резидуалне везе праве озбиљну разлику у погледу оптимизације, штавише, обука дубоке неуронске мреже без резидуалних веза је неупоредиво тежа.

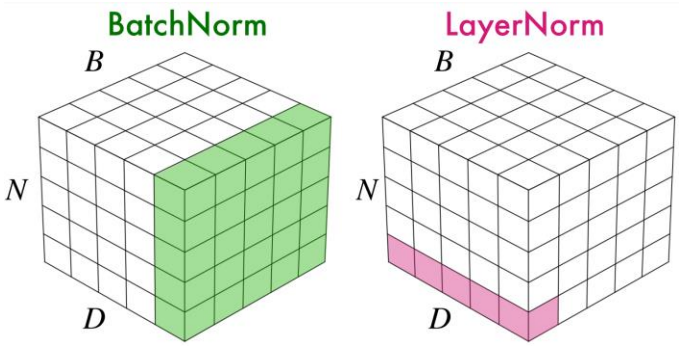
Такође, постоје једна веома често цитирана интуиција која објашњава зашто резидуалне везе помажу процесу оптимизације. Ова интуиција се базира на отклањању проблема нестајућих градијената (енг. *vanishing gradients*). Ово је проблем до којег долази услед великог броја узастопних множења матрица и као такав, дуго је представљао несавладиву препреку у домену дубоког учења.

Поред резидуалних веза, тема овог потпоглавља су и подблокови који се тичу нормализације слојева (енг. *Layer Normalization*). Ови подблокови су на слици 3.1.3.1 назначени са *“Norm”*. Нормализација у контексту дубоког учења представља процес којим се врши стандардизација улазног слоја подешавањем и скалирањем активација (тренутних вредности неурона).

Вероватно најчешће коришћени тип нормализације је ткзв. беч нормализација (енг. *Batch Normalization*), међутим у контексту архитектуре визуелних трансформера, користи се нормализација слојева (*Layer Normalization*). Ова два приступа су, заправо, веома слична. У оба случаја врши се нормализација улаза одузимањем од његове средње вредности ($\mathrm{E}[x]$) и дељењем квадратним кореном збира варијансе и позитивне константе која се користи због стабилности (ε). Овако добијени резултат се затим скалира вредношћу означеном са γ, на шта се додаје научени *bias*. Ово објашњење се односи на формулу 3.1.3.4.3 која се налази испод.

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{Var[x] + \varepsilon}} \cdot \gamma + \beta \quad (3.1.3.4.3)$$

Разлика између беч нормализације и нормализације слојева огледа се у начину израчунавања средње вредности и варијансе улаза. Наиме, до сада је било приче о улазној матрици која је димензије $N \times D$, при чему је N број улазних печева + 1 (класни токен), а D димензионалност вектора над којом трансформер енкодер ради. Ипак, у реалности, због оптимизационих разлога заправо се процесирају 3D тензори који су величине $B \times N \times D$, при чему B представља величину беча. На слици испод се налази визуелизација овог тензора и начин обраде у случају беч нормализације и у случају нормализације по слојевима.



Слика 3.1.3.4.1 Нормализација по бечевима наспрам нормализација по слојевима

У беч нормализацији, средња вредност и варијанса се рачунају дуж димензије самог беча, док се у случају нормализације по слојевима ово рачуна дуж димензије *embedding* вектора. Ово практично значи да, нормализација по слојевима не поседује никакву зависност са димензијом беча, тако да предикција не зависи директно од композиције бечева. Такође, још један додатни бенефит у односу на беч нормализацију је коришћење идентичне процедуре и у процесу тренирања и у процесу тестирања, као и могућност паралелизације овог процеса.

Овим потпоглављем се заокружује прича око архитектуре и начина рада *ViT* модела, иницијалног трансформер модела у овој области. Као што је раније у раду споменуто, овај модел поседује одређене проблеме у перформансама, с обзиром на то да га описује квадратна сложеност (апроксимативно), као и то да је потребан поприлично велики скуп података над којим би овај модел извршио претренирање како би класификација била могућа. Разлог због којег је потребан велики скуп података за претренирање лежи у генерализацији овог модела, који за разлику од конволуционих мрежа не поседује *inductive bias* у довољној мери. Као што се и могло претпоставити, у наредним годинама, након изласка оригиналног рада на ову тему, појављују се модели који потенцијално превазилазе ове проблеме на одређене начине. У наредном поглављу биће речи о овим моделима.

3.2 *DeiT* (*Data-efficient image Transformer*) архитектура

У последњих неколико година, област рачунарског вида доживела је револуцију појавом основног *ViT* модела, описаног у претходном поглављу. Овај модел, користи све принципе које се користе и у области обраде природних језика, са изузетком адаптације на област рачунарског вида и класификације слика. Међутим, ограничења као што су потреба за обимном рачунарском снагом и великим скуповима података ометала су широко усвајање ових трансформера, посебно у сценаријима са ограниченим ресурсима. Ово поглавље истражује иновативно решење предложено радом [8], наглашавајући практичност тренирања трансформер архитектуре.

Vision Transformer (*ViT*) је отворио пут за коришћење трансформера у рачунарском виду, демонстрирајући своје способности на масивном скупу података од 300 милиона узорака. Овај скуп података је интерни у компанији *Google*, те није јавно доступан. Са друге стране, *DeiT* је трениран над добро познатим и јавно доступним *ImageNet* скупом

података који је 10 пута мањи од претходно поменутог. Самим тим, у раду се наводи да је временски период који је потребан за тренирање овог модела само два до три дана на једној машини са четири или осам графичких картица.

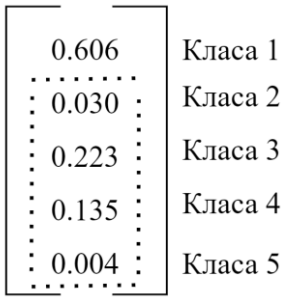
Како би *DeiT* архитектура и начин рада били потпуно јасни, неопходно је појаснити следеће концепте:

1) **Дестилација знања** (енг. *Knowledge distillation*) – Преноси знање са једног модела на други.
2) **Аугментација** – Креирање различитих варијанти улаза.
3) **Регуларизација** – Смањење претренираности модела (енг. *Overfitting*)

3.2.1 Дестилација знања

Дестилација знања се као концепт први пут појављује у раду [9]. Ова техника се користи за тренирање мањег, студент модела (енг. *Student model*) у циљу имитације понашања споријег, већег, али тачнијег, учитељ модела (енг. *Teacher model*). Потреба за овим се огледа у немогућности широке примене модела који су велики (пример су трансформер архитектуре), а притом, овакви модели су све чешћи, с обзиром на убрзани развој вештачке интелигенције и технологије уопште.

Знање које се преноси са учитељ модела се назива тамно знање (енг. *Dark knowledge*). Ова врста знања је представљена излазним вероватноћама класа неког модела које не одговарају предиктованој класи (класи са највећом вероватноћом). Да би ово било јасније, биће илустровано примером. Нека је излаз из неуронске мреже (излаз добијен провлачењем кроз коначну *softmax* функцију) следећи:

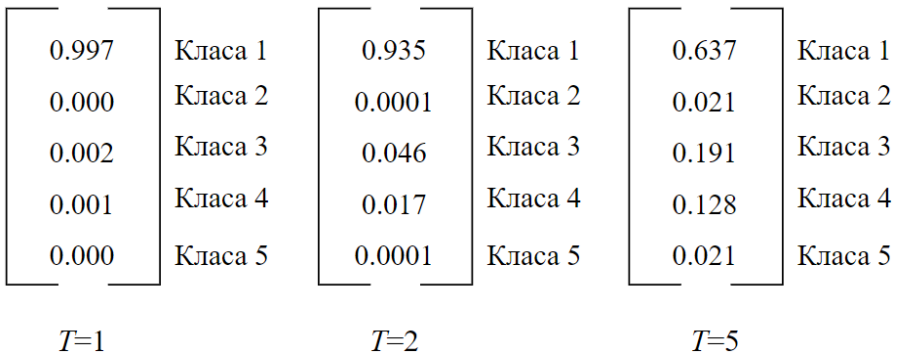


Слика 3.2.1.1 Излаз из softmax функције

Тамно знање је оно што се налази у тачкастом оквиру. То знање је веома битно када „начин размишљања“ једног модела желимо да пренесемо на други модел. Међутим, коришћење стандардног приступа за *softmax* функцију је такав да у великој већини случајева, вредности вероватноћа имају малу ентропију. Ово значи да највећа вредност буде приближна јединици, а остале буду много близу нули, тј. без икаквих информација. Како би се ово превазишло, користи се фактор температуре у контексту *softmax* функције. Ова модификација *softmax* функције дата је у наставку.

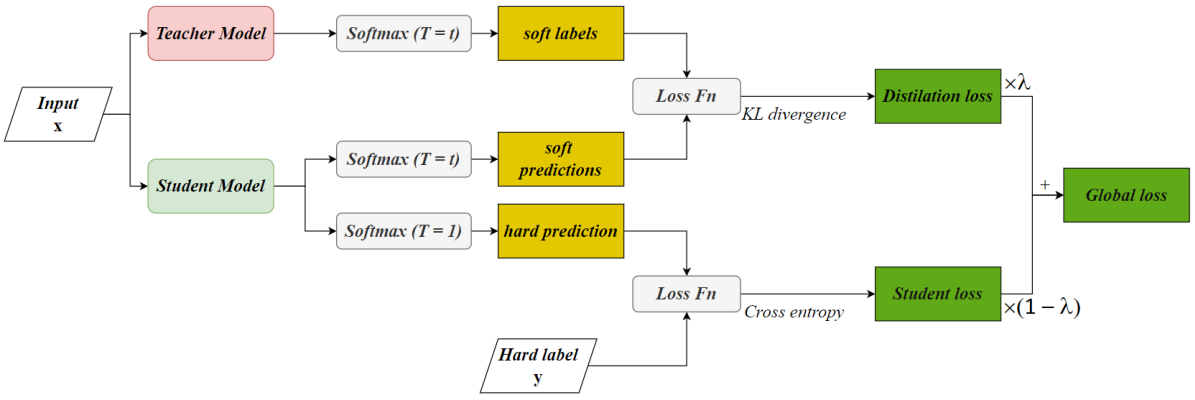
$$softmax(x) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \qquad (3.2.1.1)$$

Константа T која је присутна у овом изразу се назива „Температура“. Она потпомаже „пеглању“ дистрибуције вероватноћа које добијамо на излазу. Стандардна *softmax* функција је иста као и ова, осим што важи T=1. Са повећањем ове вредности, формира се „мекша“ дистрибуција вероватноћа (енг. *softer probability distribution*) по класама. Због овога, ствара се већа ентропија (нечистоћа) у подацима, тј. долази до смањивања високих вредности, а повећавања ниских, на начин да редослед ових вредности по величини остаје исти. На овај начин, тамно знање учитељ модела постаје богатије информацијама о граници одлуке коју је научио за сваки улаз. Илустрација овог концпета је дата на слици 3.2.2.



Слика 3.2.1.2 Утицај температурног фактора на *softmax*

На слици видимо, како, на пример, класа 2 из вредности вероватноће 0.000 са температуром T=1 расте у вредност 0.021 у ситуацији када је T=5. На овај начин се моделу који конзумира ове податке (студент моделу) прослеђује више информација о томе како учитељ модел заправо „размишља“. У наставку следи опис процеса дестилације, који је дат на слици 3.2.1.3.



Слика 3.2.1.3 Процес дестилације знања

Улазни подаци *x* се прослеђују и учитељ и студент моделу. Посматрајмо најпре учитељ модел. Након добијеног излаза, потребно је добити коначан резултат у виду вероватноћа. Ово добијамо коришћењем *softmax* функције у оквиру које користимо параметар температуре *T=t*, који је у овом случају већи од 1. На овај начин, добијамо такозване меке лабеле (енг. *soft labels*). Ове лабеле суштински представљају предикцију модела учитеља.

Сада, посматрајући студент модел, над његовим излазом се примењују две независне *softmax* функције. Прва функција има параметар *T* за који важи *T*>1, а друга нема тај параметар, тј. важи да је *T*=1. Излази из прве *softmax* функције носе назив меке предикције (енг. *soft predictions*), а излаз из друге носи назив тврде предикције (енг. *hard predictions*).

Након што се добије *soft predictions*, врши се рачунање функције губитка (енг. *loss function*) коришћењем приступа који се назива *Kullback-Leivler* (*KL*) *divergence* (релативна ентропија), означеног са *KL*(*p, q*). Ова функција губитка је у контексту дестилације знања добила назив дестилациони губитак (енг. *distillation loss*). Назив јој је овакав јер се односи на губитак који је резултат поређења меканих лабела добијених из учитељ модела и меканих предикција које су добијене из студент модела. Следи формула по којој се рачуна наведени губитак.

$$KL(p,q) = \sum_i p_i(x) log \frac{p_i(x)}{q_i(x)} \quad (3.2.1.2)$$

У овој формули $p_i(x)$ представља вероватноћу којом се *x* појављује у дистрибуцији вероватноћа *p* (излаз из одговарајуће *softmax* функције). Слично, $q_i(x)$ је вероватноћа којом се *x* појављује у дистрибуцији вероватноћа *q* (излаз из одговарајуће *softmax* функције). Овако добијени *loss* се затим множи са T^2 што у овом случају представља нормализациони фактор.

$$L_{KD} = T^2 KL \quad (3.2.1.3)$$

Са друге стране, посматрајући слику, приметно је да се између тврдих предикција и тврдих лабела (енг. *hard label*), које представљају тачне вредности, рачуна друга функција губитка која се назива *student loss*. Овај губитак се рачуна, за разлику од претходног, коришћењем унакрсне ентропије (енг. *cross entropy*).

$$\mathcal{L}_{CE} = -\sum_{x \in classes} p(x) \log q(x) \quad (3.2.1.4)$$

Након израчунавања оба губитка, коришћењем фактора λ, а потом и међусобним сабирањем добијамо глобални губитак (енг. *global loss*) који касније игра кључну улогу у пропагацији уназад кроз студент модел. На овај начин је могуће пренети знање из великог модела који је претрениран на великим скуповима података на мањи, једноставнији модел. Овај приступ носи назив мека дестилација (енг. *soft distillation*). [8]

$$\mathcal{L}_{global} = (1-\lambda)\mathcal{L}_{CE} + \lambda L_{KD} \quad (3.2.1.5)$$

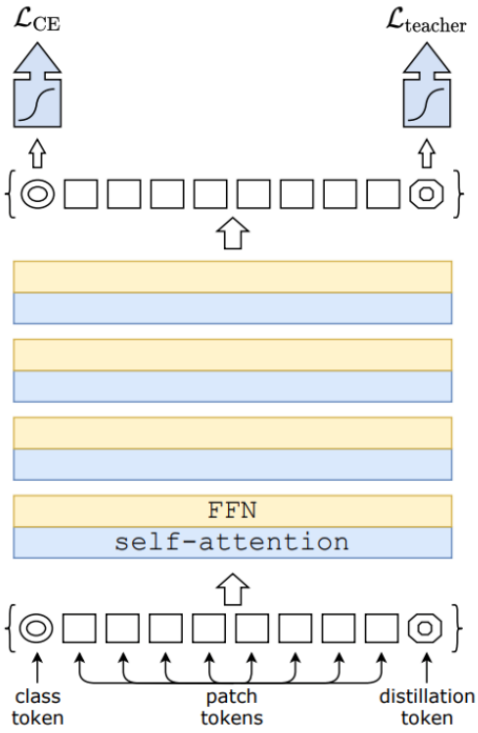
Нека су Z_s и Z_t излази из студент, односно учитељ модела, пре примене *softmax* алгоритма. Симбол ψ описује *softmax* функцију, *y* представља стварне вредности (*ground truth*), а *T* температуру *softmax* функције. Након разлагања формуле 3.2.1.5, добијамо коначну формулу приказану у наставку. [8]

$$\mathcal{L}_{global} = (1-\lambda)\mathcal{L}_{CE}(\psi\,(Z_s), y) + \lambda T^2 KL(\psi\,(Z_s/T), \psi(Z_t/T)) \qquad (3.2.1.6)$$

У раду који се тиче *DeiT* архитектуре, предлаже се коришћење модификоване верзије приступа дестилацији знања у односу на ону која је управо описана. Овај, модификовани приступ, носи назив тврда дестилација (енг. *hard label distillation*). У овом случају, тврда лабела (енг. *hard label*) учитеља, назначена са y_t, игра исту улогу као и стварна лабела. Наиме, излаз из учитељ модела пролази кроз *softmax* функцију чија је вредност температуре *T*=1. Ово значи да се дословно узима предиктована лабела учитеља и да се користи на исти начин као и стварна лабела (енг. *ground truth label*). [8]

$$\mathcal{L}_{global}^{hardDistill} = \frac{1}{2}\mathcal{L}_{CE}(\psi\,(Z_s), y) + \frac{1}{2}\mathcal{L}_{CE}(\psi\,(Z_s), y_t) \qquad (3.2.1.6)$$

Битно је напоменути да се за учитељ модел *DeiT* архитектуре користи једна од најсавременијих конволуционих неуронских мрежа. Ова мрежа је претходно обучена на *ImageNet* скупу података. Архитектура студент модела, за разлику од учитеља, представља благу модификацију архитектуре визуелног трансформера која је детаљно описана у претходном поглављу. Још једна од битних измена у односу на претходно описану дестилацију знања је прослеђивање излаза из конволуционе неуронске мреже (учитељ модела) улазу у трансформер архитектуру. На слици испод, налази се се *DeiT* архитектура из оригиналног рада. [8]



Слика 3.2.1.4 *DeiT* архитектура

Као што се види на слици, токени класе и токени печева се прослеђују трансформер енкодеру као што је то био случај и код *ViT* архитектуре. Међутим, у овом случају прослеђује се и вектор који се назива дестилациони токен (енг. *distillation token*) који, заједно са осталим токенима пролази кроз све блокове трансформер енкодера (блокове само-пажње и блокове вишеслојних перцептрона).

Овај токен је веома сличан класном токену, с тим што за циљ има да репродукује тврду лабелу (предиктовану од стране учитељ модела), уместо стварне лабеле. Ово значи да дестилациони токен учи из излаза учитељ модела (рачуна функцију губитка у односу на предикцију конволуционе неуронске мреже), као што је то случај и у оригиналној дестилацији знања, док са друге стране остаје комплементаран (допуњујући, али не исти) класном токену. Оно што се даље напомиње у овом раду је велики утицај управо овог дестилационог токена на побољшање у перформансама трансформера. Следи формула по којој се рачуна укупна функција губитка у случају *DeiT* трансформер архитектуре. На основу ове функције се касније врши пропагације уназад.

$$L = \frac{1}{2}L_{CE} + \frac{1}{2}L_{teacher} \tag{3.2.1.7}$$

3.2.2 Аугментација и регуларизација

Аугментација података је кључна техника која омогућава повећање разноликости тренинг скупа путем вештачког проширивања доступних података. Ова техника не само да

помаже у превазилажењу проблема са ограниченим скупом података, већ и доприноси бољој генерализацији модела. С обзиром на то да је *DeiT* трансформер модел, потребан му је велики скуп података, те се у великој мери ослања на аугментацију.

Регуларизација, с друге стране, представља кључан елемент у борби против преприлагођавања модела на тренинг скуп. Различите технике регуларизације, као што су *dropout*, *L*1 и *L2* регуларизација, играју кључну улогу у одржавању равнотеже између високе прецизности на тренинг скупу и способности модела да генерализује на невиђеним подацима.

У контексту *DeiT* архитектуре, коришћене су разне технике како би се постигло најефикасније и најбоље решење када је класификација слика у питању. У раду се наводи и табела која описује добијене резултате коришћењем различитих техника аугментације и регуларизације. На слици испод, налази се споменута табела. [8]

Ablation on ↓	Pre-training	Fine-tuning	Rand-Augment	AutoAug	Mixup	CutMix	Erasing	Stoch. Depth	Repeated Aug.	Dropout	Exp. Moving Avg.	top-1 accuracy: pre-trained 224^2	top-1 accuracy: fine-tuned 384^2
none: DeiT-B	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✗	81.8 ±0.2	83.1 ±0.1
optimizer	SGD	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✗	74.5	77.3
	adamw	SGD	✓	✗	✓	✓	✓	✓	✓	✗	✗	81.8	83.1
data augmentation	adamw	adamw	✗	✗	✓	✓	✓	✓	✓	✗	✗	79.6	80.4
	adamw	adamw	✗	✓	✓	✓	✓	✓	✓	✗	✗	81.2	81.9
	adamw	adamw	✓	✗	✗	✓	✓	✓	✓	✗	✗	78.7	79.8
	adamw	adamw	✓	✗	✓	✗	✓	✓	✓	✗	✗	80.0	80.6
	adamw	adamw	✓	✗	✗	✗	✓	✓	✓	✗	✗	75.8	76.7
regularization	adamw	adamw	✓	✗	✓	✓	✗	✓	✓	✗	✗	4.3*	0.1
	adamw	adamw	✓	✗	✓	✓	✓	✗	✓	✗	✗	3.4*	0.1
	adamw	adamw	✓	✗	✓	✓	✓	✓	✗	✗	✗	76.5	77.4
	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✓	✗	81.3	83.1
	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✓	81.9	83.1

Слика 3.2.2.1 Табела резултата

Као што се може видети, коришћено је доста различитих техника и комбинација за аугментацију и регуларизацију и самим тим, добијени су различити резултати. Ови резултати поприлично варирају са променом аугментационих и регуларизационих техника.

На овај начин, завршава се потпоглавље које се тиче *DeiT* архитектуре. Као што се могло и претпоставити, ова архитектура нуди озбиљан напредак у перформансама у поређењу са претходном *ViT* архитектуром. Користи се доста мањи скуп података и мањи трансформер модел. Међутим, не треба заборавити да је учитељ модел (*CNN*) ипак трениран пре трансформер модела, те је и за то ипак потребна велика рачунарска моћ иако се у овом случају користи само као готов модел. У наставку следи прича о још једном виду оптимизације иницијалне *ViT* архитектуре. У питању је *Swin Transformer*.

3.3 *Swin Transformer* архитектура

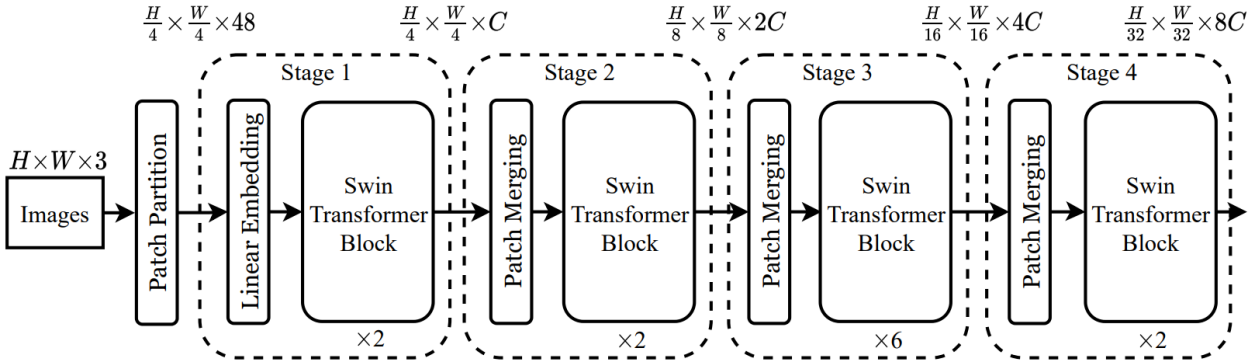
До сада су у раду представљене две трансформер архитектуре које су намењене области рачунарског вида, основна архитектура визуелног трансформера (*ViT*) и њена модификација која је донела доста доброг у погледу перформанси (*DeiT*). У овом поглављу биће речи о још једној трансформер архитектури која носи назив *Swin Transformer* архитектура.

У области рачунарског вида, као што је и раније истакнуто, примарну улогу играју конволуционе неуронске мреже. Један од разлога због којег су оне и даље међу најпопуларнијима у овој области је њихова универзалност. Ово значи да постоје индивидуалне конволуционе неуронске мреже које су способне да самостално решавају више различитих задатака у области рачунарског вида (класификацију слика, семантичку сегментацију слика, детекцију објеката, синтезу слика и др.). Само неке од конволуционих неуронских мрежа које су способне да реше све ове проблеме су *VGG, ResNET, DenseNet* итд.

У случају архитектуре визуелног трансформера, ово до сада није било могуће. Један од разлога је фиксна величина печа. Поред овога, за величину печа се не препоручује да буде димензија мањих од 16x16 пиксела. Разлог овоме је меморијска сложеност механизма само-пажње. Као што је раније напоменуто, у питању је квадратна сложеност у односу на број печева слике. С обзиром на то да са смањењем величине печа, расте број печева, ово аутоматски значи да оваква архитектура визуелног трансформера није погодна за задатке попут семантичке сегментације слика, где је основна идеја одредити лабелу (класу) сваког пиксела на слици. Семантичка сегментација слика је само један од примера где, до сада описане, трансформер архитектуре није могуће применити. Поред овога, детекција објеката је такође проблем, с обзиром на различита скалирања (величине објеката на сликама).

Истраживачи из компаније *Microsoft* су препознали овај проблем и, сходно томе, издали рад [10] у коме су презентовали свој хијерархијски трансформер модел. Главни циљ ове архитектуре је постављање нове универзалне основне структуре у области рачунарског вида (енг. *general-purpose backbone for computer vision*), тј. креирање универзалног модела који ће, по узору на поједине конволуционе неуронске мреже, моћи да се користи за све задатке у области рачунарског вида.

Начин на који ради ова трансформер архитектура се заснива на креирању такозваних хијерархијских мапа. Ове мапе су заправо печеви који повећавају своје димензије проласком кроз слојеве овог модела. Такође, битна функционалност коју *Swin transformer* архитектура поседује (по којој је и добила име) су померајући прозори (енг. *shifted windows*). Овај приступ омогућава већу ефикасност, ограничавајући рачунање само-пажње на непреклапајуће локалне прозоре, притом дозвољавајући и одређену комуникацију између различитих прозора. На овај начин, меморијска сложеност у односу на број печева се смањује са квадратне на линеарну, што представља изврстан напредак. На слици 3.3.1 дат је преглед архитектуре овог трансформер модела. У наредним секцијама ће сваки индивидуални сегмент бити посебно анализиран.



Слика 3.3.1 Swin transformer архитектура [10]

3.3.1 *Patch Partition* и *Linear Embedding* блокови

Као што се може видети на слици 3.3.1, на улазу у трансформер архитектуру налази се слика величине $H\times W\times 3$. Прва ствар која се примењује над овом сликом јесте њена трансформација у печеве величине 4×4. Одмах се да приметити, да су ови печеви поприлично мањи од оних које је користила *ViT* архитектура (16×16) што је у контексту *ViT* архитектуре било практично немогуће због непостојања графичких картица са довољном меморијом која би испратила раније поменуте меморијске квадратне сложености механизма само-пажње. Међутим, аутори овог модела су смислили начин да ову квадратну сложеност успешно сведу на линеарну. [10]

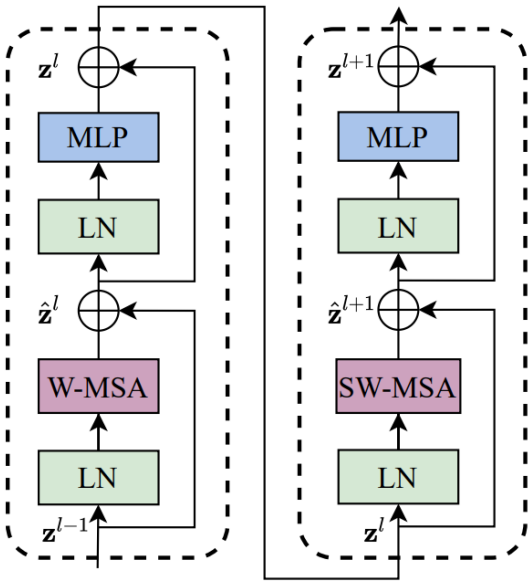
Посматрањем *Patch Partition* блока види се примена једноставне трансформације облика улазног тензора из $H\times W\times 3$ у $\frac{H}{4}\times\frac{W}{4}\times 48$. Интуитивно гледано, ово представља трансформацију пиксела у печеве. Пиксели у овом случају садрже три канала (*R*, *G* и *B*), док печеви садрже 48 канала. Ово значи да је сваки печ заправо вектор чија је димензија 48. Међутим, по узору на универзалне конволуционе неуронске мреже, потребно је додатно повећавати број канала проласком кроз дубље слојеве.

Због овога, постоји блок назван *Linear Embedding* који врши мапирање из 48 канала у *C* канала, где притом важи $C>48$. Један од једноставнијих начина на који је могуће применити овај *Linear Embedding* је коришћењем потпуно повезане мреже. Ова мрежа би вршила мапирање сваког печа у *C* димензиони вектор. Међутим, с обзиром на то да је кôд *Swin transformer* модела јавно доступан, могуће је видети да су ова два блока заправо спојена у један блок назван *Patch Embedding.*

У овом хибридном блоку користи се конволуциони кернел величине 4×4 са кораком (енг. *stride*) 4. На овај начин сваки кернел заправо прихвата читав печ као улаз. С обзиром на то да је величина корака 4, не долази ни до каквог преклапања ових кернела. Ово практично значи да се од 4×4 пиксела на улазној слици креира један излазни печ који поседује само један канал на излазу. Наравно, у циљу формирања *C* канала потребно је применити *C* кернела над улазом.

3.3.2 *Swin transformer* блок

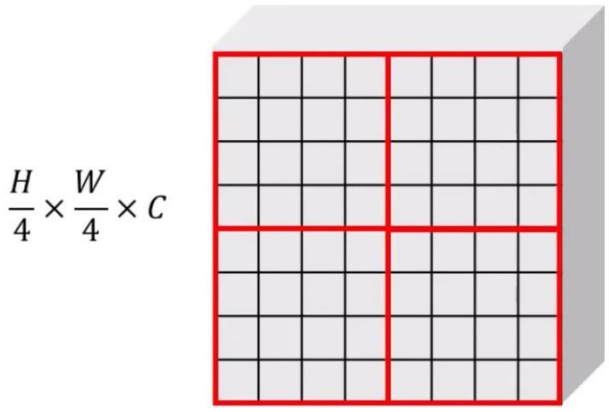
Претходно обрађени блокови су по много чему слични одговарајућем делу оригиналне трансформер архитектуре. Главни допринос рада [10] огледа се у предлагању новог типа блока који носи назив *Swin transformer* блок. Овај блок се састоји од више консеквентних трансформер блокова који су присутни у паровима (видети на слици 3.3.2.1), те је број ових подблокова увек дељив бројем два.



Слика 3.3.2.1 *Swin Transformer* блок [10]

Као што се може видети на слици 3.3.2.1, ови блокови изгледају поптупуно исто као и трансформер енкодер блокови у оригиналном *ViT* транформеру, са присуством мале измене. Наиме, уместо *Multi-Head Attention* слоја користе се одређене алтернативе. У првом блоку се налази *W-MSA* слој (*Window based Muli-Self Attention Layer*), а у другом *SW-MSA* слој (*Shifted Window Multi-Self Attention Layer*). Као што је већ речено, након сваког *W-MSA* слоја, неопходно је имати и *SW-MSA* слој, те због овога број ових блокова мора бити дељив бројем два.

Први од ова два блока је веома једноставан за разумевање. Наиме, потребно је дефинисати такозвани прозор (енг. *window*) који представља својеврсну матрицу печева, као што је приказано на слици 3.3.2.2.

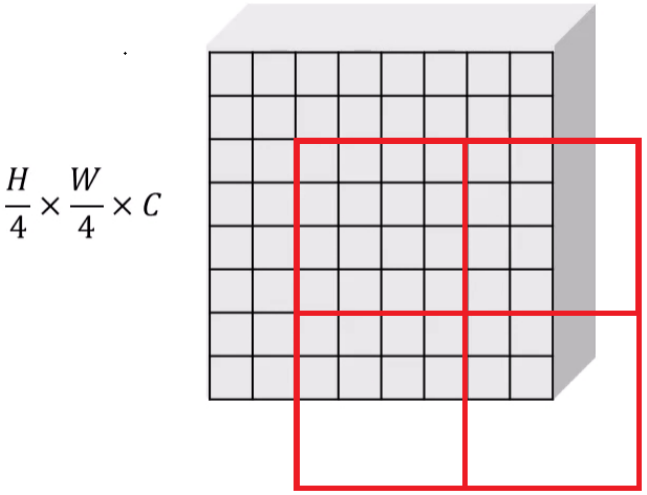


Слика 3.3.2.2 Прозори састављени од печева

У овом случају, матрица печева је величине 4×4. Битно је напоменути и то да се ови прозори међусобно не преклапају. Разлог због којег се формирају ови прозори је постојање *W-MSA* слоја. Због његовог постојања ће се, уместо провлачења свих печева одједном кроз овај слој, провући сви печеви истог прозора. На овај начин, у овом конкретном примеру ће бити формирана четири излаза из *W-MSA* слоја која одговарају сваком од појединачних прозора. Као што се може и претпоставити, управо је ово принцип по коме се врши смањење раније поменуте комплексности са квадратне на линеарну, јер се механизам само-пажње примењује само над печевима унутар исте групе.

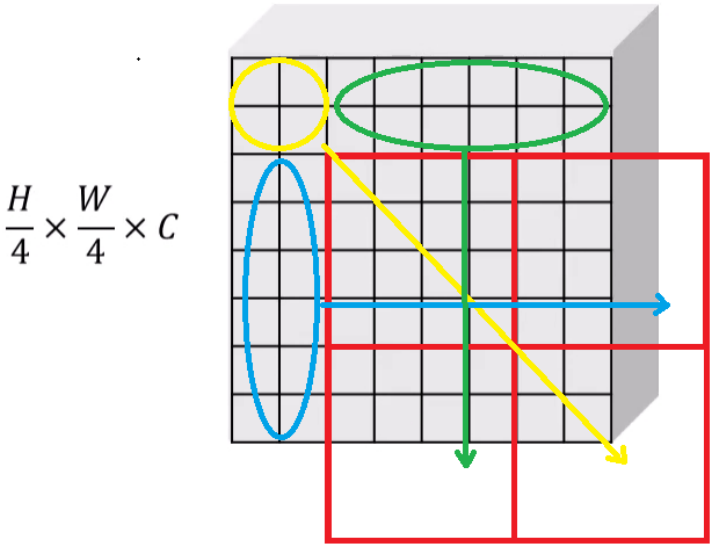
Интуитивно посматрајући, ово итекако има смисла, с обзиром на то да, на пример, горњи леви пиксел обично нема никакву везу са пикселом који се налази у доњем десном углу. Међутим, у случају када би овај блок био коришћењ самостално (без постојања *SW-MSA* блока), постојао би проблем. Рецимо да се на самој средини слике налази неки објекат. Нека тај објекат припада делимично сваком од прозора. У овом случају, слој само-пажње не би могао да схвати шта је заправо овај објекат.

У циљу решавања овог проблема, постоји још један блок назван *SW-MSA*. Објашњење које се крије иза овог блока јесте померање прозора за два печа удесно, односно надоле, након чега следи поновно израчунавање само-пажње између ових прозора. У овом тренутку то изгледа као што је приказано на слици 3.3.2.3.



Слика 3.3.2.3 Померање прозора за два печа

Као што се може видети на слици, постоји празан простор у прозору, тј. простор у оквиру ког се не налази нити један печ. Наивно решење би било попунити овај простор нулама, међутим, овај простор у том случају не би имао никакве информације. Софистицираније решење, које се примењује у овом раду се назива циклично померање (енг.*cycled shifting*). Илустрација овог решења се налази на слици 3.3.2.4.

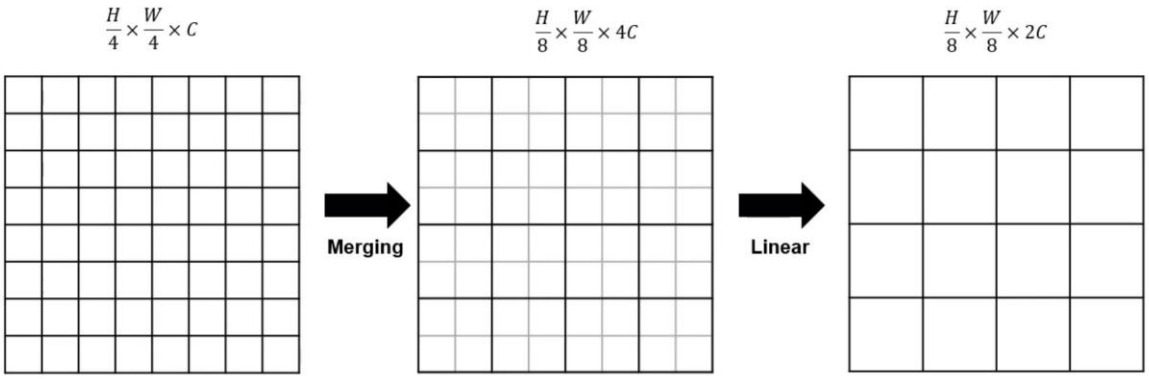


Слика 3.3.2.4 Циклично померање

Оваквим приступом решава се проблем детекције било ког објекта на било којој позицији унутар слике. Постоји још само један проблем. Тај проблем представља третирање појединих печева као суседа иако то заправо нису. Узимимо за пример печеве уокривене у плаво. Након цикличног померања, десиће се да постају суседи са печевима са десног краја слике. У раду се напомиње да је за решење овог проблема коришћена такозвана *Masked MSA*, у чије детаље имплементације даље није залажено.

3.3.3 Блок спајања печева

Последњи блок који је потребно описати у циљу потпуног разумевања ове архитектуре је блок чији је назив блок спајања печева (енг. *patch merging block*). Оно што је главна функционалност овог блока је, као што му и само име каже, спајање суседних печева у један већи печ. Иницијално, величина печа је била 4×4, међутим, након спајања, добијамо печеве величине 8×8. Наравно, као последица мењања облика тензора, број канала се такође повећава 4 пута. Дакле, 2 пута се печ смањује по ширини, 2 пута по дужини, што даје укупно смањење од 4 пута у погледу просторне резолуције. Због овога се број канала увећава 4 пута како би се задржала одговарајућа димензија. Након тога, долази до проласка овог тензора кроз још један линеарни слој, како би се мапирала димензија из 4C у 2C, при чему величина печева остаје иста. На слици 3.3.3.1 илустрован је овај поступак.



Слика 3.3.3.1 Поступак спајања печева

Као што се може видети на слици 3.3.1, објашњени поступци у овим поглављима се даље понављају кроз фазе (енг. *stage*) при чему се повећава број печева и на тај начин покривају објекти на сликама који могу бити веома мали до веома велики. С обзиром на то да је раније споменуто да се ови трансформери могу користити како у класификацији слика, тако и у сегментацији слика и детекцији објеката, следи укратко објашњење како ово постићи. Како би се постигла класификација слика, довољно је узети излаз из последњег трансформер блока и пропустити га кроз линеарни слој, а потом применити *softmax* функцију. Са друге стране, за потребе детекције објеката и сегментације слика, потребно је узети излазе из свих слојева и користити их као фичере, што је заправо врло слично приступу који преовладава у конволуционим неуронским мрежама.

За крај овог поглавља, следи табела која врши упоређивање свих трансформер архитектура које су описане у овом раду. Битно је напоменути да су, у овом случају, модели тренирани над *ImageNet-1K* скупом података. Ово представља лошу погодност по оригинални *ViT* трансформер јер његова суштина лежи у генерализацији и тренингу над огромним скуповима података, за разлику од друга два модела које је могуће тренирати и на мањим скуповима. У табели се јасно види да, у овом случају (врло битно нагласити), победу односи *Swin* трансформер и у погледу искоришћене меморије и у погледу тачности. Ова табела, приказана је на слици 3.3.3.2.

Regular ImageNet-1K trained models					
method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
ViT-B/16 [19]	384^2	86M	55.4G	85.9	77.9
ViT-L/16 [19]	384^2	307M	190.7G	27.3	76.5
DeiT-S [57]	224^2	22M	4.6G	940.4	79.8
DeiT-B [57]	224^2	86M	17.5G	292.3	81.8
DeiT-B [57]	384^2	86M	55.4G	85.9	83.1
Swin-T	224^2	29M	4.5G	755.2	81.3
Swin-S	224^2	50M	8.7G	436.9	83.0
Swin-B	224^2	88M	15.4G	278.1	83.5
Swin-B	384^2	88M	47.0G	84.7	84.5

Слика 3.3.3.2 Упоредна табела резултата над *ImageNet-1K* скупом података [10]

4 Практични део мастер рада

До сада, главна тема овог рада била је теоријско објашњење визуелних трансформера. Ово је укључивало темељну обраду начина рада и примене сваке од споменутих трансформер архитектура.

Тема овог поглавља бави се практичним делом рада, тј. укључује примену наведених трансформер архитектура над конкретним скупом података (о самом скупу података биће речи у наредним потпоглављима). У ову сврху коришћена је техника пренесног учења (енг. *Transfer learning*) ради постизања релевантнијих и бољих резултата, међутим, у циљу темељнијег истраживања, поред пренесеног учења, укључена је и имплементације *ViT* архитектуре. На крају, извршена је упоредна анализа свих описаних трансформер архитектура која је довела до финалних закључака.

4.1 Скуп података

Скуп података који се користи у овом истраживању састоји се од слика водених инсеката. Слике су димензија 512x512 пиксела и обухватају широк спектар различитих врста инсеката, приказујући њихове карактеристичне особине и детаље. Са чак 100 различитих класа инсеката, овај скуп података представља изазов за класификацију, посебно због сличности између појединих врста које су често суптилне и тешко уочљиве људском оку.

Воденим инсектима је својствена комплексна морфологија и варијација у бојама и узорцима, што чини њихову класификацију изузетно захтевном. Међутим, ови инсекти имају кључну улогу у екосистемима слатких вода, те је њихова прецизна идентификација од виталног значаја за очување природе. На слици 4.1.1 приказани су неки од узорака овог скупа података.

Слика 4.1.1 Примери слика из скупа података

С обзиром на велики број класа и сличност између појединих инсеката, одлучено је да се за решавање проблема класификације користи техника пренесеног учења, уз имплементацију *ViT* архитектуре ради упоређивања перформанси.

Скуп података је организован у три групе, односно подфолдера:

1. ***train_512***: Овај подфолдер садржи 8603 инстанце које се користе за тренинг модела.
2. ***val_512***: Валидациони скуп са 3366 инстанци омогућава евалуацију перформанси модела током процеса обуке и подешавања параметара.
3. ***test_512***: Тестни скуп садржи 5138 инстанци које се користе за коначну евалуацију перформанси модела.

Укупно, овај скуп података садржи 17107 инстанци, што обезбеђује довољно података за обуку, валидацију и тестирање модела. Наведени скуп података пружа изазовно окружење за тестирање перформанси различитих трансформер архитектура.

Наредна поглавља ће детаљније истражити примену пренесеног учења и *ViT* архитектуре на овом скупу података, као и резултате добијене кроз упоредну анализу перформанси различитих модела.

4.2 *ViT base* модел и коришћење пренесеног учења

ViT base модел је основни трансформер модел из рада [4]. Овај модел се састоји из 12 трансформер енкодер блокова, при чему је радна димензија овог трансформера D=768. Укупно, модел има преко 86 милиона параметара.

Приликом претренирања овог модела (енг. *pretraining*), тј. саме иницијалне обуке, како се наводи у раду [4] коришћен је *Adam* оптимизатор са параметрима $\beta_1 = 0.9$ и $\beta_2 = 0.999$. У раду се наглашава блага предност овог оптимизатора у односу на *SGD* (*Stochastic gradient descent*) који је дао нешто лошије резултате. *Adam* оптимизатор је популаран алгоритам оптимизације који ефикасно ради са великим бројем параметара и адаптивно прилагођава стопу учења (енг. *learning rate*) за сваки параметар. Такође, у раду се даље наводи, коришћена је и техника регуларизације звана пропадање тежина (енг. *weight decay*) која је постављена на вредност 0.1. Ова техника представља метод регуларизације који смањује величину тежина током тренинга како би се спречило пренаучавање модела.

Претренирање овог модела, као и других модела који ће бити описани у наставку рада, спроведено је над популарним и раније споменутим *ImageNet* скупом података. *ImageNet* је један од највећих и најчешће коришћених скупова података у области рачунарског вида и као такав садржи преко милион различитих слика подељених у хиљаду класа.

У коду 4.2.1 може се видети начин на који је наведени модел, заједно са својим претренираним тежинама, учитан.

```
# 1. Учитавање тежина претренираног ViT base модела
pretrained_vit_weights = torchvision.models.ViT_B_16_Weights.DEFAULT
# 2. Учитавање ViT base модела са одговарајућим тежинама
pretrained_vit = torchvision.models.vit_b_16(weights=pretrained_vit_weights).to(device)
# 3. Замрзавање свих слојева
for parameter in pretrained_vit.parameters():
parameter.requires_grad = False

# 4. Промена главе класификатора и њено одмрзавање
class_names = train_data.classes
pretrained_vit.heads = nn.Linear(in_features=768, out_features=len(class_names)).to(device)
```

Код 4.2.1 Учитавање претренираног модела

4.2.1 Процес тренирања и евалуације модела

Након учитавања претренираног модела, извршено је тренирање, тј. фино подешавање готовог модела (енг. *fine tuning*). За почетак, извршено је замрзавање свих унутрашњих слојева трансформер архитектуре изузев последњег слоја (Код 4.2.1.1), тј. главе вишеслојног перцептрона. На овај начин, експлицитно се омогућава тренинг искључиво последњег слоја, без било каквог мењања вредности тежина у претходним слојевима. Када је извршено замрзавање слојева започет је процес тренирања модела.

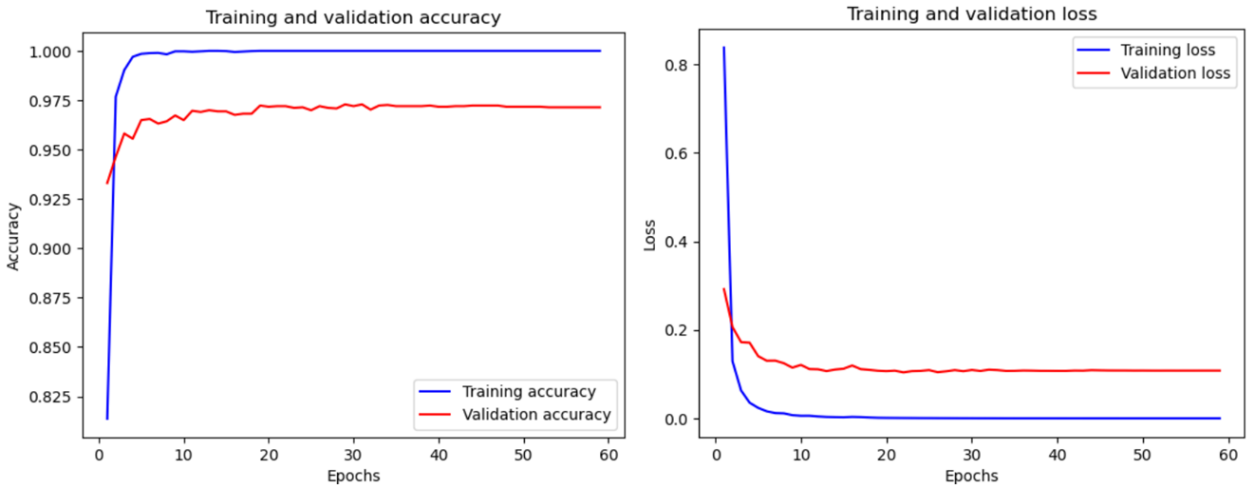
За потребе тренирања модела коришћене су препоруке из рада [4]. Наиме, као што је то био случај и код претренирања, коришћен је *Adam* оптимизатор са стопом учења од 3e-3, али и унакрсна ентропија као функција губитка (енг. *cross-entropy loss*). Модел је пуштен на обучавање кроз 60 епоха са стрпљењем од 15 епоха (енг. *early stopping*), ово практично значи да би у случају непобољшавања валдиационе тачности у трајању од 15 епоха, модел престао са даљом обуком. Поред овога, коришћено је и стрпљење у погледу стопе учења које је износило 12 епоха. На овај начин би се стопа учења смањила 10 пута у случају непостојања напретка током 12 епоха. Ово је могуће видети у коду 4.2.1.1.

```
# Креирање оптимизатора и функције губитка
optimizer = torch.optim.Adam(params=pretrained_vit.parameters(), lr=3e-3)
loss_fn = torch.nn.CrossEntropyLoss()
# Тренирање главе класификатора претренираног ViT модела
set_seeds(33)
pretrained_vit_results, model = engine.train(device=device, model=pretrained_vit,
                                train_dataloader=train_dataloader_pretrained,
                                val_dataloader=val_dataloader_pretrained,
                                optimizer=optimizer, loss_fn=loss_fn, epochs=60, patience=15,
                                lr_patience=12,best_model_path='Results/Models/BestModel.pth')
```

Код 4.2.1.1 Подешавање параметара и тренирање модела

Тренирање је извршено на графичкој картици *Nvidia GTX 1050ti* са 2*GB RAM* меморије, што је резултирало трајањем тренинга од преко 10 сати због обимности овог модела и самог скупа података. Током тренинга, постигнути су следећи резултати:

Валидациона тачност (тачност над валидационим скупом података) достигла је максималну вредност од 97.23%, при чему је *early stopping* активиран након 34 епохе због непостојања напретка. На слици испод, налазе се криве валидационог губитка и валидационе тачности кроз епохе.



4.2.1.1 Валидациони губитак и тачност кроз епохе

Након извршеног тренирања, кроз добијени модел су пуштени тестни подаци како би се дошло до коначног решења када је у питању овај скуп података. Резултат за тачност над тестним скупом пдоатака износи 97.35%, док губитак износи 0.1005. Овај процес омогућио је добијање релевантних и прецизних резултата, употребљавајући пренесено учење, чиме је додатно унапређено разумевање и практична примена ове архитектуре у области рачунарског вида.

4.3 Имплементација *ViT base* модела

Након коришћења пренесеног учења, у даљем истраживању, применила се имплементација *ViT base* трансформер модела коришћењем *PyTorch* библиотеке. Ова имплементација биће главна тема овог поглавља заједно са резултатима који су добијени над, раније споменутим, скупом података који се тиче водених инсеката. С обзиром на то да је у овом раду већ темељно објашњена архитектура овог модела, у овом поглављу ће бити подразумевано познавање одређених термина и концепата.

Као што је већ објашњено, *ViT* архитектура најпре врши претварање улазне слике у секвенцу печева (креирањем печева, додавањем позиционе информације, итд.), а затим, тако модификоване печеве провлачи кроз низ трансформер енкодер блокова. Сходно овоме је и извршена имплементација. Наиме, имлементација обухвата класе које представљају основне компоненте *ViT* модела, укључујући следеће:

1. ***PatchEmbedding*** – претвара улазну слику у *patch embedding* векторе.
2. ***MutiheadSelfAttentionBlock*** – примењује нормализацију и вишеглавну самопажњу на улазне векторе.
3. ***MLPBlock*** – креирају се два линеарна слоја са *GELU* активационом функцијом.
4. ***TransformerEncoderBlock*** – креира се блок енкодера који се састоји од блока вишеглавне самошажње и блока вишеслојног перцептрона, са додатним резидуалним везама за обе компоненте.
5. ***ViT*** – врши интеграцију свих претходних модула и креира одговарајући визуелни трансформер.

Имплементације свих ових класа заједно са описним коментарима, могуће је видети у секцији „Прилог 1“ овог рада.

4.3.1 Процес тренирања и евалуације модела

Након имплементације *ViT* модела и процеса тренирања, резултати су далеко од очекивања, с обзиром на претходни модел који је користио пренесено учење. Очекивано је било да ће перформансе бити знатно лошије, имајући у виду да је суштина визуелних трансформера у претходном обучавању над обимним скуповима података. Ипак, ови резултати пружају увид у изазове и ограничења директног тренирања *ViT* модела без претходно научених тежина.

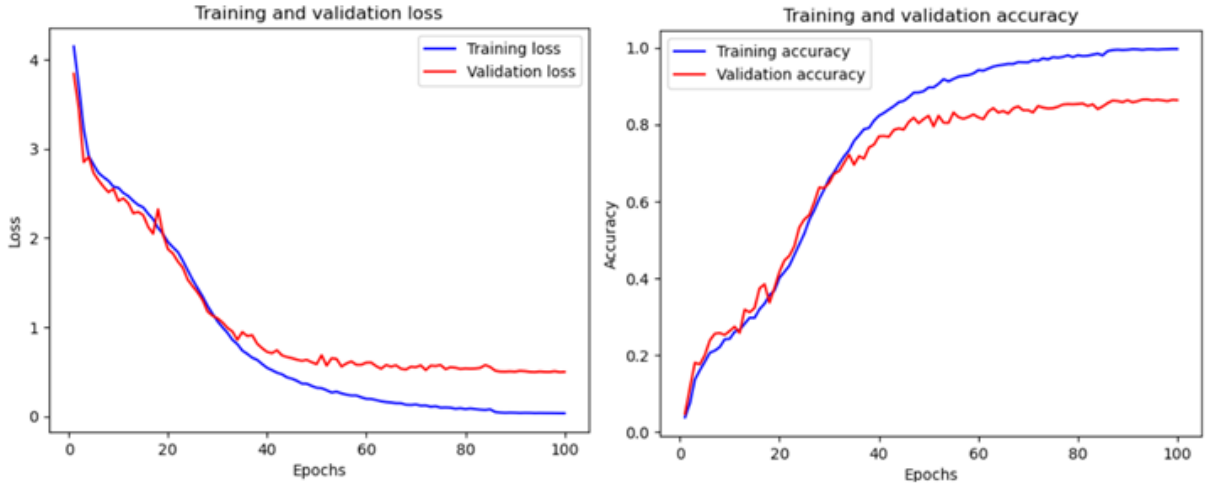
Приликом тренирања, испробано је неколико различитих стратегија. Иницијално, коришћене су препоруке из оригиналног рада, које укључују, између осталог, коришћење *Adam* оптимизатора. Међутим, приметно је било да је *Adam* оптимизатор доводио до проблема с нестајањем градијента, што је резултирало преурањеним завршетком процеса тренирања већ након седам епоха. Након дубљег истраживања, закључено је да би могло бити ефикасније користити *SGD* као алгоритам оптимизације, што је довело до бољих резултата. Код везан за подешавања параметара и тренирања модела дат је у наставку.

```
optimizer = torch.optim.SGD(vit.parameters(), lr=1e-3, momentum=0.9)
# Подешавање функције губитка
loss_fn = torch.nn.CrossEntropyLoss()
# Set the seeds
set_seeds(33)
# Тренирање модела и чување резултата
results, model = engine.train(model=vit, train_dataloader=train_dataloader,
                      val_dataloader=val_dataloader, optimizer=optimizer,
                      loss_fn=loss_fn, epochs=100, patience=20, lr_patience=12,
                      best_model_path='Results/Models/best_model_implementation.pth',
device=device)
```

Код 4.3.1.1 Подешавање параметара и тренирање модела

Важно је напоменути да се сада модел тренира у потпуности, а не само глава као у претходном моделу који је користио пренесено учење. Ова промена приступа, наравно, захтева више времена за обуку. Сам процес тренирања трајао је више од 16 сати на раније поменутој графичкој картици.

Тренирање је, као што се види у коду 4.2.1.1, вршено кроз 100 епоха, где се *early stopping* извршава након 20 епоха без побољшања, а смањење стопе учења се врши након 12 неуспелих епоха. Резултати који су добијени овом приликом су далеко бољи од иницијалног очекивања. Наиме, тренирање је трајало свих 100 епоха, где притом није дошло ни до једног смањења стопе учења. Најбоља валидациона тачност која је постигнута износи 86.52%, а најбољи валидациони губитак износи 0.4945. С обзиром на то да је модел и креиран и трениран од нуле, ови резултати надмашују сва очекивања. Међутим, ваља приметити да су резултати и даље далеко лошији од оних постигнутих пренесеним учењем, где је тачност на валидационом скупу била 97.23%. На слици испод, налазе се одговарајуће криве.



Слика 4.3.1.1 Валидациони губитак и тачност кроз епохе

Као што се може приметити на слици 4.3.1.1, учење је поприлично нестабилно. Наиме, током целог процеса учења постоје мали успони и падови, да би се на крају, у последњих 20 епоха модел примирио. Међутим, осим ове нестабилности, нема трагова пренаучености модела, с обзиром на то да криве које се тичу тренинг скупа података градацијски расту (падају) упоредо са валидационим кривама.

Када је реч о тестирању, модел је показао солидне перформансе, с губитком од 0.4938 и тачношћу од 87.19% на тестном скупу података. Иако су ови резултати прихватљиви, још увек су знатно инфериорни у поређењу са претходним моделом који је користио пренесено учење.

Узимајући у обзир ове резултате, јасно је да директно тренирање *ViT* модела може бити изазовно, посебно у смислу постизања перформанси које се могу поредити са моделом који је обучен над *ImageNet* скупом и који користи пренесено учење. Ипак, ови

резултати указују на могућност побољшања путем даљег експериментисања с различитим стратегијама тренирања, хиперпараметрима и архитектуром модела.

4.4 *Swin base* модел и коришћење пренесеног учења

Swin base модел је модел визуелног трансформера представљен у раду [10]. Овај конкретни модел је највећи од свих модела који су представљени у раду, а да су притом тренирани на *ImageNet* скупу података који садржи 1000 различитих класа.

Како се у раду [10] наводи, модел је обучаван коришћењем *AdamW* оптимизатора са 300 епоха и косинусним опадањем стопе учења, уз 20 епоха линеарног загревања. *AdamW* оптимизатор се разликује од *Adam* оптимизатора по томе што укључује фактор регуларизације тежина. Ова промена може помоћи у стабилизацији процеса тренирања, посебно у случајевима када се јављају проблеми са конвергенцијом или пренаглашеним учењем. Величина беча, тј. број узорака који се користе у једној итерацији, био је постављен на 1024, док је почетна стопа учења износила 0.001, уз *weight decay* параметар постављен на 0.05. Током тренинга, наводи се у даљем тексту, примењене су различите технике аугментације и регуларизације, изузев, раније споменуте, понављајуће аугментације како би се побољшала генерализација модела.

У коду 4.4.1 може се видети начин на који је наведени модел, заједно са својим претренираним тежинама, учитан.

```
# 1. Учитавање тежина претренираног swin base модела
pretrained_swin_weights = torchvision.models.Swin_B_Weights.DEFAULT

# 2. Учитавање модела и додела учитаних тежина
pretrained_swin = torchvision.models.swin_s(weights=pretrained_swin_weights).to(device)

# 3. Замрзавање свих слојева унутар модела
for parameter in pretrained_swin.parameters():
    parameter.requires_grad = False

# 4. Креирање главе класификатора и њено одмрзавање
class_names = train_data.classes
set_seeds(33)
pretrained_swin.head = nn.Linear(in_features=768, out_features=len(class_names)).to(device)
```

Код 4.4.1 Учитавање претренираног модела

4.4.1 Процес тренирања и евалуације модела

Након учитавања претренираног модела, извршено тренирање, тј. фино подешавање готовог модела (енг. *fine tuning*). На самом почетку, извршено је замрзавање свих слојева унутар трансформер архитектуре изузев последњег слоја (Код 4.2.1), тј. главе вишеслојног

перцептрона (енг. *MLP head*). На овај начин, експлицитно се омогућава тренинг искључиво последњег слоја, без било каквог мењања вредности тежина у претходним слојевима. Када је извршено замрзавање слојева започет је процес тренирања модела.

Као и код финог подешавања *ViT* модела, и овде су коришћени идентични параметри. Дакле, тренинг је вршен 60 епоха, са стопом учења од 0.003, активацијом раног завршавања након 15 неуспелих епоха и смањењем стопе учења у случају 12 епоха без икаквог напретка у погледу валдиационе тачности. Такође, битно је напоменути да је приликом финог подешавања коришћен класичан *Adam* оптимизатор, а не *AdamW* како је саветовано у раду [10] због давања приметно бољих резултата. Приликом коришћења *AdamW* оптимизатора јављао се и проблем нестајућих градијената као што је то био случај са имплементираним *ViT* моделом. Као и тренинзи претодних модела, и овај модел је трениран на раније споменутој графичкој картици. Тренинг овог модела је трајао нешто мање од 9 сати.

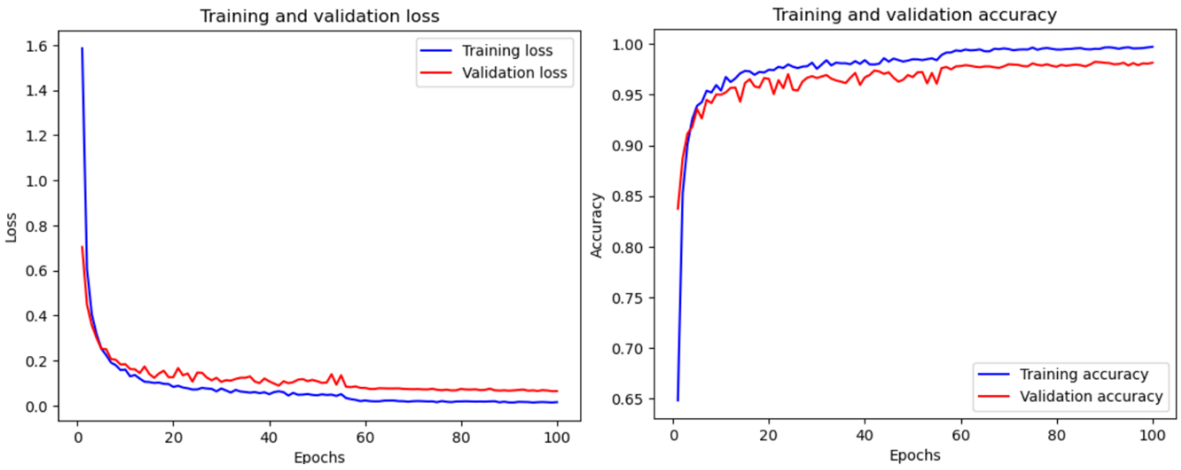
На слици испод, налазе се добијени резултати приликом тренирања модела у последњих 5 епоха.

```
Epoch: 56 | train_loss: 0.0346 | train_acc: 0.9891 | val_loss: 0.0849 | val_acc: 0.9761 | learning_rate: 0.000300
Epoch: 57 | train_loss: 0.0294 | train_acc: 0.9915 | val_loss: 0.0826 | val_acc: 0.9773 | learning_rate: 0.000300
Epoch: 58 | train_loss: 0.0263 | train_acc: 0.9916 | val_loss: 0.0857 | val_acc: 0.9749 | learning_rate: 0.000300
Epoch: 59 | train_loss: 0.0207 | train_acc: 0.9936 | val_loss: 0.0794 | val_acc: 0.9779 | learning_rate: 0.000300
Epoch: 60 | train_loss: 0.0235 | train_acc: 0.9930 | val_loss: 0.0792 | val_acc: 0.9782 | learning_rate: 0.000300
```

Слика 4.4.1.1 Последњих 5 епоха приликом тренирања

Као што се може видети на слици, валидациона тачност модела се и даље повећавала те је у последњој (60. епохи) добијена највећа тачност (97.82%), али и најмањи валидациони губитак (0.0792). Сходно овоме, долази се до закључка да наставак тренирања овог модела има смисла и да би се на тај начин добили још бољи резултати.

Уследило је дотрениравање постојећег модела на још 40 епоха (укупно 100 епоха). Резултати који су добијени су и очекивани. Наиме, дошло се до достизања најбољег резултата када је валидациона тачност у питању, и то 98.23%, док је забележени губитак био 0.0672. На слици која следи налазе се графови валидационе тачности и валидационих губитака који су достизани током тренинга.



Слика 4.4.1.2 Валидациони губитак и тачност кроз епохе

На слици се могу видети мала одступања, али је тренирање поприлично уједначено све време, без присуства пренаучавања модела.

Након извршеног тренирања, кроз добијени модел су пуштени тестни подаци како би се дошло до коначног решења када је у питању споменути скуп података. Резултат за тачност над тестним скупом пдоатака износи 98.14%, док губитак износи 0.0724. Овај процес омогућио је добијање најбољох резултата у поређењу са свим другим добијеним резултатима.

4.5 *DeiT base* модел и коришћење пренесеног учења

У овом поглављу детаљно је описан *DeiT base* модел који је коришћен у истраживању, заједно са техником пренесеног учења која је примењена како би се побољшале перформансе модела.

DeiT base модел који се користи је модел из рада [8] са величином печа 16x16, што значи да обрађује слике у блоковима величине 16x16 пиксела. Овај модел је заправо варијација, раније испитаног, *ViT-B* модела, при чему су параметри фиксирани на $D = 768$, а број глава самопажње је 12. Осим тога, у раду су представљене и мање варијанте овог модела попут *DeiT-S* (*small*) и *DeiT–Ti* (*tiny*), чије су карактеристике детаљно описане у оригиналном раду.

У процесу обуке овог модела, у раду се даље наводи, коришћен је учитељ модел назван *RegNetY-16GF*, конволуциона мрежа са 84 милиона параметара, која је такође тренирана на истим подацима и аугментацијама као и сам *DeiT* модел. Овај учитељ модел постиже значајну тачност од 82.9% на *ImageNet* скупу података. Тренирање *DeiT base* модела извршено је у трајању од 300 епоха, где се притом наводи да су се са повећањем броја епоха добијали бољи резултати.

У коду 4.5.1 може се видети начин на који је наведени модел, заједно са својим претрениранim тежинама, учитан.

```
# 1. Учитавање модела и претренираних тежина
pretrained_DeiT = torch.hub.load('facebookresearch/DeiT:main', 'DeiT_base_patch16_224',
pretrained=True).to(device)
# 2. Замрзавање свих слојева модела
for parameter in pretrained_DeiT.parameters():
    parameter.requires_grad = False
class_names = train_data.classes
set_seeds(33)
n_inputs = pretrained_DeiT.head.in_features
print(n_inputs)
# 3. Креирање нове главе класификатора
pretrained_DeiT.head = nn.Linear(in_features=n_inputs,out_features=len(class_names)).to(device)
```

Код 4.5.1 Учитавање претренираног модела

4.5.1 Процес тренирања и евалуација модела

Као што се може видети у коду 4.5.1, за разлику од претходног начина учитавања модела, коришћењем *torchvision.models* модула, овде се директно учитава оригинални модел компаније *Facebook*. Разлог овоме је непостојање *DeiT* архитектуре у оквиру овог модула.

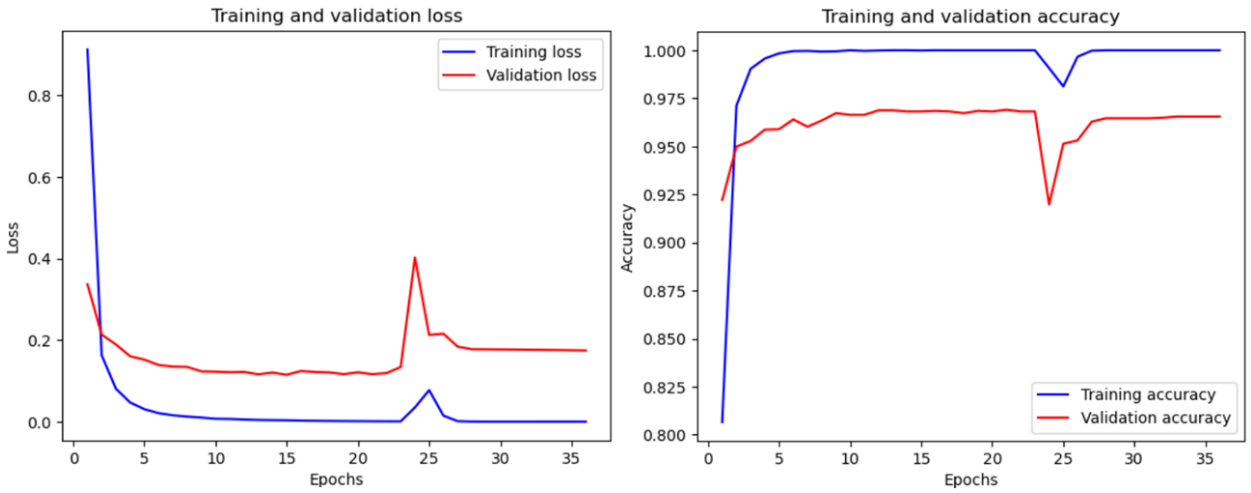
Приликом финог подешавања сви слојеви, изузев последњег, су замрзнути, што омогућава да се само последњи слој прилагоди специфичним карактеристикама новог скупа података. Као оптимизатор је коришћен *Adam* оптимизатор, како је и препоручено у оригиналном раду [8]. Остали хиперпараметри који се тичу тренирања су потпуно идентични свим претходним моделима, те неће бити додатно појашњени. Као и раније, тренинг модела извршен је на графичкој картици *GTX 1050ti*, а трајао је око 8 сати.

Током процеса тренинга, праћени су губитак и тачност модела на тренинг и валидационом скупу. На слици 4.5.1.1 из лога може се видети да су губитак и тачност постепено скоро потпуно престали са напредовањем, а да је притом тренинг тачност достигла вредност 1. Ово значи да је наступила пренаученост модела и да даље тренирање не води бољим резултатима. На крају, коначна максимална вредност валидационе тачности достигла је стабилну вредност од 96.55% након неколико епоха, а претходно поменуто ненапредовање довело је до обуставе обуке због недостатка побољшања у тачности током последњих 15 епоха.

```
Epoch: 28 | train_loss: 0.0005 | train_acc: 1.0000 | val_loss: 0.1777 | val_acc: 0.9646 | learning_rate: 0.003000
Epoch: 29 | train_loss: 0.0004 | train_acc: 1.0000 | val_loss: 0.1774 | val_acc: 0.9646 | learning_rate: 0.000300
Epoch: 30 | train_loss: 0.0004 | train_acc: 1.0000 | val_loss: 0.1771 | val_acc: 0.9646 | learning_rate: 0.000300
Epoch: 31 | train_loss: 0.0004 | train_acc: 1.0000 | val_loss: 0.1768 | val_acc: 0.9646 | learning_rate: 0.000300
Epoch: 32 | train_loss: 0.0004 | train_acc: 1.0000 | val_loss: 0.1764 | val_acc: 0.9649 | learning_rate: 0.000300
Epoch: 33 | train_loss: 0.0004 | train_acc: 1.0000 | val_loss: 0.1760 | val_acc: 0.9655 | learning_rate: 0.000300
Epoch: 34 | train_loss: 0.0004 | train_acc: 1.0000 | val_loss: 0.1757 | val_acc: 0.9655 | learning_rate: 0.000300
Epoch: 35 | train_loss: 0.0003 | train_acc: 1.0000 | val_loss: 0.1751 | val_acc: 0.9655 | learning_rate: 0.000300
Epoch: 36 | train_loss: 0.0003 | train_acc: 1.0000 | val_loss: 0.1746 | val_acc: 0.9655 | learning_rate: 0.000300
```

Слика 4.5.1.1 Последње епохе приликом финог подешавања модела

На слици 4.5.1.2 налазе се графикони који прате валидациону тачност и валидациони губитак током читавог тренирања.



Слика 4.5.1.2 Валидациони губитак и тачност кроз епохе

Као што се са слике може приметити, присутна је пренаученост модела, и непостојање напретка већ од 5. епохе. Такође, може се приметити и велико одступање и губитка и тачности у епохи 24. Разлог овоме није познат.

Након извршеног финог подешавања и тренирања, кроз добијени модел су пуштени тестни подаци како би се дошло до коначног решења када је у питању споменути скуп података. Резултат за тачност над тестним скупом података износи 96.89%, док губитак износи 0.1079. Очекивања, када је у питању овај модел, нису адекватно испуњена с обзиром на то да су добијени најлошији резултати, ако се изузме имплементирани *ViT* модел.

4.6 Упоредна анализа

У овом поглављу биће спроведена упоредна анализа коришћених трансформер модела и добијених резултата. Фокус ће бити на њиховим перформансама у контексту раније описаног скупа података који се тиче водених инсеката, Поред тога, биће објашњен појам индуктивне пристрасности (енг. *inductive bias*), као и њен утицај у свакој од обрађених

трансформер архитектура. Кроз ову анализу, циљ је створити дубље разумевање предности и недостатака сваке архитектуре и како се оне међусобно разликују у погледу перформанси и флексибилности. На крају, ова упоредна анализа требало би да пружи корисне увиде у избор одговарајуће архитектуре трансформера у зависности од специфичних захтева задатака и карактеристика скупа података.

У табели 4.6.1 дат је збирни приказ свих добијених резултата као и коришћених хиперпараметара приликом процеса тренирања модела, где су, притом, за све моделе коришћене исте стопе учења (0.003), *early stopping* (15) и смањење стопе учења након 12 неуспелих епоха (изузев имплементације *ViT* модела где се стопа смањивала након 20 неуспелих епоха).

	Укупно епоха	**Оптимизатор**	**Најбоља валидациона тачност**	**Тачност на тестном скупу**
ViT base – пренесено учење	60	*Adam*	97.23%	97.35%
ViT base – имплементација	100	*SGD*	86.52%	87.19%
Swin base - пренесено учење	100	*Adam*	98.14%	98.14%
DeiT base – пренесено учење	60	*Adam*	96.55%	96.89%

Табела 4.6.1 Коначни резултати модела

Као што се из табеле 4.6.1 може видети, најбољи резултат је постигнут коришћењем *Swin base* трансформера. На први поглед, чини се да је главни фактор овог успеха већи број епоха. Међутим, важно је напоменути да је *ViT base* достигао свој лимит пре 60. епохе, док је, са друге стране, *Swin* трансформер у тој фази постигао најбољи резултат, што је мотивисало додатних 40 епоха обуке. Како би разлог за овакве резултате био јаснији, потребно је укратко објаснити појам **индуктивне пристрасности** (енг. *inductive bias*).

Индуктивна пристрасност обухвата скуп претпоставки и предиспозиција које алгоритам учења (у овом случају, трансформер модел) користи за правилне предикције. У суштини, то је претходно стечено знање на које се алгоритам ослања приликом генерализације тренинг података на тестне податке. Сваки алгоритам машинског и дубоког учења има одређени степен индуктивне пристрасности, било експлицитне или имплицитне.

У овом конкретном случају, најмању индуктивну пристрасност поседује иницијални визуелни трансформер (*ViT*). Разлог томе је жеља за максимизацијом генерализације. На пример, конволуционе неуронске мреже имају изражену индуктивну пристрасност због постојања конволуција и филтера, што унапред даје алгоритму информацију да блиски пиксели имају везе једни с другима. С друге стране, треба споменути и неуронске мреже које скоро да немају никакву индуктивну пристрасност, попут *ANN* мрежа, где су неурони распоређени у слојевима без претходних претпоставки о обради специфичног скупа података.

С обзиром на то да *ViT* има најмању индуктивну пристрасност, било је очекивано да ће овај модел имати лошије резултате, посебно модел који је имплементиран и трениран од нуле. Разлог за оваква очекивања лежи у недовољно великом скупу података потребном за темељну обуку овако великог модела. Премда је имплементирани *ViT* модел испунио очекивања, *ViT* који је користио пренесено учење их је и надмашио, показујући се бољим од *DeiT* архитектуре са истим бројем параметара.

DeiT архитектура, по правилу, има највећу индуктивну пристрасност, те је очекивано да ће дати најбоље резултате. Међутим, то није био случај, што јасно указује да није могуће једнозначно посматрати резултате кроз призму индуктивне пристрасности, иако је то нешто што се прво узима у обзир. Индуктивна пристрасност *DeiT* архитектуре је велика због коришћења учитељског модела који је у овом случају конволуциона неуронска мрежа. Очекивање бољих резултата од основне *ViT* архитектуре проистиче из тога што је *DeiT* у основи исти модел са додатком велике конволуционе неуронске мреже.

На крају, модел који се најбоље показао над овим скупом података је *Swin base* модел. С обзиром на то да је *Swin* архитектура најновија од свих и карактерише је пре свега ефикасност у погледу перформанси у поређењу са *ViT* архитектуром, овај резултат је итекако прихватљив. Што се тиче индуктивне пристрасности, присутна је у мањој мери него што је то случај код *DeiT* трансформера. У овом случају, индуктивна пристрасност се огледа у динамичности блокова и примени механизма самопажње само између суседних блокова уместо између свих што је био случај код ранијих трансформер архитектура.

5 Закључак

У овом раду детаљно је анализирана еволуција класификације слика, почевши од раних вештачких неуронских мрежа до савремених модела визуелних трансформера. Приказана је генеза конволуционих неуронских мрежа, са посебним освртом на пионирске моделе попут *LeNet*-5 и *AlexNet*, који су поставили темеље за напредак у овој области. Развој конволуционих мрежа новијег доба и појава трансформер архитектуре додатно су унапредили способност машина да "виде" и "разумеју" садржај слика на начин који је раније био незамислив.

Посебно је истакнут утицај оригиналне трансформер архитектуре на рачунарски вид, који је директно проузроковао стварање иницијалног визуелног трансформера за потребе области рачунарског вида. Поменути модел се први пут појавио у научном раду компаније *Google* [4]. Овај рад се својим највећим делом детаљно бави управо овом архитектуром која представља револуционарни приступ у класификацији слика. *ViT* архитектура, са својим модулима за креирање и трансформацију печева, додавање позиционе информације и трансформер енкодером, демонстрира напредак у способности машина да ефикасно процесуирају визуелне податке.

Анализа *Single-Head* и *Multi-Head Attention* механизама, као и блокова вишеслојног перцептрона унутар *ViT* архитектуре, даје могућност за темељно разумевања ових концепата и указује на значајне иновације које омогућавају моделу да постигне изванредне резултате у класификацији слика. Такође, у раду се спомиње и принцип генерализације који овај модел нуди, а који је од огромне важности. Непостојање *inductive-bias* концепта, тј. усмеравања модела да учи на унапред дефинисани начин, представља велики корак у поређењу са конволуционим неуронским мрежама. Из овога се може закључити да је за потребе *ViT* архитектуре потребан велики скуп података.

Поред тога, рад разматра и *DeiT* (*Data-efficient Image Transformer*) архитектуру, која унапређује *ViT* приступ кроз дестилацију знања и технике аугментације и регуларизације. Дестилација знања је у овом поглављу описана детаљно, с обзиром на њен значај у области дубоког учења и вештачке интелигенције. Ово омогућава моделу да одржи високе перформансе чак и када су подаци ограничени, што је од суштинског значаја за практичну примену.

Swin Transformer архитектура је такође представљена као важан допринос у пољу рачунарског вида. *Swin Transformer*, као један од најновијих доприноса, представља значајан корак напред у смислу скалабилности и ефикасности, нудећи модуларни приступ који омогућава фину прилагодљивост модела специфичним потребама примене у области рачунарског вида. Идеја овог модела је била креирати такозвану универзалну основну структуру у области рачунарског вида (енг. *general-purpose backbone for computer vision*). Сходно овоме, циљ овог модела је било адекватно решење за задатке попут сегментације слика, детекције објеката и класификације слика.

У последњем поглављу овог мастер рада детаљно је представљен практични део истраживања. Описани су коришћени трансформер модели, скуп података и резултати експеримента. Анализом добијених резултата, уочено је да *Swin base* трансформер модел даје најбоље перформансе, што се делимично може приписати већој индуктивној пристрасности у поређењу са другим моделима, изузев *DeiT* модела који није испунио очекивања.

У закључку, *ViT* архитектура и њени деривати представљају прекретницу у рачунарском виду, нудећи нове методе за обраду и анализу визуелних информација. Њихова примена има потенцијал да трансформише многе индустрије, омогућавајући развој интелигентнијих и ефикаснијих система за рачунарски вид. Даљи развој и истраживање у овој области ће вероватно довести до још напреднијих модела, који ће даље побољшати способност машина да интерпретирају визуелне податке. Очекује се да ће *ViT* архитектура и њени наследници наставити да играју кључну улогу у будућим иновацијама у рачунарском виду.

Литература

[1] *How Artificial Intelligence Revolutionized Computer Vision: A Brief History*, доступно на: *https://www.motionmetrics.com/how-artificial-intelligence-revolutionized-computer-vision-a-brief-history/* (приступљено 21.02.2024)

[2] *Convolutional neural networks:1998-2023 overview*, доступно на: *https://www.superannotate.com/blog/guide-to-convolutional-neural-networks* (приступљено 20.02.2024)

[3] *Ashish Vaswani, Noam Shazeer, Niki Parmar* и други, ***Attention Is All You Need***, 2017

[4] *Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov* и други, ***An Image is Worth 16x16 Words:Transformers for Image Recognition at Scale***, 2020

[5] *Vision Transformer: What It Is & How It Works [2023 Guide]*, доступно на: *https://www.v7labs.com/blog/vision-transformer-guide* (приступљено 22.02.2024)

[6] *Attention Networks: A simple way to understand Self Attention*, доступно на: *https://medium.com/@geetkal67/attention-networks-a-simple-way-to-understand-self-attention-f5fb363c736d* (приступљено 22.02.2024)

[7] *Dan Hendrycks, Kevin Gimpel*, ***Gaussian Error Linear Units (GELUs)***, 2016

[8] *Hugo Touvron, Matthieu Cord, Matthijs Douze* и други, ***Training data-efficient image transformers & distillation through attention***, 2020

[9] *Geoffrey Hinton, Oriol Vinyals, Jeff Dean,* ***Distilling the Knowledge in a Neural Network***, 2015

[10] *Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, Baining Guo*, ***Swin Transformer: Hierarchical Vision Transformer using Shifted Windows***, 2021

Прилози

Прилог 1. Изворни код имплементације *ViT* архитектуре

```
# 1. Креирање класе која је поткласа класе nn.Module
class PatchEmbedding(nn.Module):
    """Претвара 2D улазну слику у 1D patch embedding вектор
    Args:
        in_channels (int): Број канала у боји за улазне слике. Подразумевано је 3.
        patch_size (int): Величина жељених печева. Подразумевано је 16.
        embedding_dim (int): Радна димензија трансформера. Подразумевано је 16.
    """
    # 2. Иницијализација класе са одговарајућим параметрима.
    def __init__(self, in_channels:int=3, patch_size:int=16, embedding_dim:int=768):
        super().__init__()
        self.patch_size = patch_size
        # 3. Креирање слоја за претварање слике у печеве
        self.patcher = nn.Conv2d(in_channels=in_channels, out_channels=embedding_dim,
                                 kernel_size=patch_size,stride=patch_size, padding=0)
        # 4. Креирање слоја за поравнавање печева у једну димензију
        self.flatten = nn.Flatten(start_dim=2, end_dim=3)
    # 5. Дефинисање forward методе
    def forward(self, x):
        # Креирати проверу да ли су улази одговарајућих величина
        image_resolution = x.shape[-1]
        assert image_resolution % self.patch_size == 0, f"Input image size must be divisble by patch
size, image shape: {image_resolution}, patch size: {patch_size}"
        # Пролазак унапред
        x_patched = self.patcher(x)
        x_flattened = self.flatten(x_patched)
        # 6. Претварање излаза у одговарајући облик
        return x_flattened.permute(0, 2, 1)

    # 1. Креирање класе која је поткласа класе nn.Module
class MultiheadSelfAttentionBlock(nn.Module):
    """Креира multi-head self-attention блок."""
    # 2. Иницијализација класа хиперпараметрима из табеле 1 рада [4]
    def __init__(self,
                 embedding_dim:int=768, # D величина из табеле 1 за ViT-base
                 num_heads:int=12, # Главе из табеле 1 за ViT-base
                 attn_dropout:float=0):
        super().__init__()
        # 3. Креирање слоја нормализације
        self.layer_norm = nn.LayerNorm(normalized_shape=embedding_dim)
        # 4. Креирање Multi-Head Attention (MSA) слоја
```

```
        self.multihead_attn = nn.MultiheadAttention(embed_dim=embedding_dim, num_heads=num_heads,
                                                    dropout=attn_dropout, batch_first=True)
    # 5. Креирање forward методе
    def forward(self, x):
        x = self.layer_norm(x)
        attn_output, _ = self.multihead_attn(query=x, key=x, value=x,need_weights=False)
        return attn_output

# 1. Креирање класе која је потклаcа класе nn.Module
class MLPBlock(nn.Module):
    """Creates a layer normalized multilayer perceptron block ("MLP block" for short)."""
    # 2. Иницијализација класе хиперпараметрима из табеле 1 и табеле 3
    def __init__(self,
                 embedding_dim:int=768, # D димензија из табеле 1 за ViT-base
                 mlp_size:int=3072, # MLP size из табеле 1 за ViT-base
                 dropout:float=0.1): # Dropout из табеле 3 за ViT-base
        super().__init__()
        # 3. Креирање слоја нормализације
        self.layer_norm = nn.LayerNorm(normalized_shape=embedding_dim)
        # 4. Креирање вишеслојног перцептрона
        self.mlp = nn.Sequential(
            nn.Linear(in_features=embedding_dim, out_features=mlp_size),
            nn.GELU(), # "MLP користи GELU активациону функцију
            nn.Dropout(p=dropout),
            nn.Linear(in_features=mlp_size, # преузима се излаз из претходног слоја
                      out_features=embedding_dim), # враћање димензије на D
            nn.Dropout(p=dropout)
        )
    # 5. Креирање forward методе
    def forward(self, x):
        x = self.layer_norm(x)
        x = self.mlp(x)
        return x

# 1. Креирање класе која је потклаcа класе nn.Module
class TransformerEncoderBlock(nn.Module):
    """Creates a Transformer Encoder block."""
    # 2. Иницијализација класе хиперпараметрима из табеле 1 и табеле 3
    def __init__(self,
                 embedding_dim:int=768, # D димензија из табеле 1 за ViT-base
                 num_heads:int=12, # Главе из табеле 1 за ViT-base
                 mlp_size:int=3072, # MLP size из табеле 1 за ViT-base
                 mlp_dropout:float=0.1, # Вредност dropout из табеле 3
                 attn_dropout:float=0): # Вредност dropout за слојеве пажње
        super().__init__()
```

```
        # 3. Креирање MSA блока
        self.msa_block = MultiheadSelfAttentionBlock(embedding_dim=embedding_dim,
num_heads=num_heads, attn_dropout=attn_dropout)
        # 4. Креирање MLP блока
        self.mlp_block =  MLPBlock(embedding_dim=embedding_dim, mlp_size=mlp_size,
                                   dropout=mlp_dropout)
   # 5. Креирање forward методе method
    def forward(self, x):
        # 6. Креирање резудуалне везе за MSA блок (додавање улаза излазу)
        x =  self.msa_block(x) + x
        # 7. Креирање резудуалне везе за MLP блок (додавање улаза излазу)
        x = self.mlp_block(x) + x
        return x

# 1. Креирање ViT класе која је изведена из nn.Module
class ViT(nn.Module):
    """Creates a Vision Transformer architecture with ViT-base hyperparameters by default."""
    # 2. Иницијализација класе хиперпараметрима из табеле 1 и табеле 3
    def __init__(self,
                 img_size:int=224, # Величина из табеле 3 у ViT раду
                 in_channels:int=3, patch_size:int=16,
                 num_transformer_layers:int=12, # Слојеви из табеле 1 за ViT-base
                 embedding_dim:int=768, mlp_size:int=3072, num_heads:int=12,
                 attn_dropout:float=0, mlp_dropout:float=0.1, embedding_dropout:float=0.1,
                 num_classes:int=1000): # Default for ImageNet but can customize this
        super().__init__()
        # 3. Величина слике мора бити дељива са величином печа
        assert img_size % patch_size == 0, f"Image size must be divisible by patch size, image size:
{img_size}, patch size: {patch_size}."
        # 4. Израчунати број печева (height * width/patch^2)
        self.num_patches = (img_size * img_size) // patch_size**2
        # 5. Креирање learnable class embedding-а (Иде заједно са patch embedding vektorima)
        self.class_embedding = nn.Parameter(data=torch.randn(1, 1, embedding_dim),
                                            requires_grad=True)
        # 6. Креирање learnable position embedding-а
        self.position_embedding = nn.Parameter(data=torch.randn(1, self.num_patches+1,
embedding_dim),requires_grad=True)
        # 7. Креирање embedding dropout вредности
        self.embedding_dropout = nn.Dropout(p=embedding_dropout)
        # 8. Креирање patch embedding слоја
        self.patch_embedding = PatchEmbedding(in_channels=in_channels, patch_size=patch_size,
                                              embedding_dim=embedding_dim)
```

```
        # 9. Креирање трансформер енкодер блокова који се ређају један на други коришћењем
nn.Sequential())
        self.transformer_encoder =
nn.Sequential(*[TransformerEncoderBlock(embedding_dim=embedding_dim, num_heads=num_heads,
                                                                    mlp_size=mlp_size,
                                                                    mlp_dropout=mlp_dropout)
for _ in range(num_transformer_layers)])
        # 10. Креирање главе класификатора
        self.classifier = nn.Sequential(
            nn.LayerNorm(normalized_shape=embedding_dim),
            nn.Linear(in_features=embedding_dim, out_features=num_classes)
        )
    # 11. Креирање forward методе
    def forward(self, x):
        # 12. Узети величину бача
        batch_size = x.shape[0]
        # 13. Креирање класног токена и његово проширење да се поклопи са величином бача
        class_token = self.class_embedding.expand(batch_size, -1, -1)
        # 14. Креирање patch embedding вектора
        x = self.patch_embedding(x)
        # 15. Конкатенирање класног токена и patch embedding вектора
        x = torch.cat((class_token, x), dim=1)
        # 16. Додавање position embedding вектора patch embedding вектору
        x = self.position_embedding + x
        # 17. Пролазак кроз embedding dropout
        x = self.embedding_dropout(x)
        # 18. Пролазак patch, position и class embedding вектора кроз трансформер енкодер слојеве
        x = self.transformer_encoder(x)
        x = self.classifier(x[:, 0]) # покреће се над свим индивидуалним бечевима на индексу 0
        return x
```