Универзитет у Београду
Машински и Математички
факултет Индустрија 4.0

Семинарски рад
Knowledge Distillation

Рачунарска интелигенција

Анђела Живановић 4010/2023
Вук Антов 4014/2024

Београд, новембар 2025. године

Sadržaj

1 Увод

Са развојем дубоког учења (енг. Deep Learning), неуронске мреже су постале кључна технологија за решавање комплексних проблема у областима као што су рачунарство слике, препознавање говора и обрада природног језика. Међутим, високе перформансе ових модела обично захтевају огромне ресурсе – милионе параметара, дугачке процесе тренирања и скупу хардверску инфраструктуру. Ова чињеница ограничава њихову примену у реалним, ресурсно ограниченим окружењима као што су мобилни уређаји, ИоТ системи или уграђене апликације.

Како би се превазишао јаз између перформанси и ефикасности, развијене су технике компресије модела, а једна од најпознатијих и најефикаснијих међу њима јесте Knowledge Distillation (KD) – пренос знања са већег модела („учитеља“) на мањи („ученик“)[1].

Концепт Knowledge Distillation први су формално представили Hinton, Vinyals и Dean (2015), који су показали да се знање великог модела може сажети у мањи модел помоћу меких ознака (енг. soft targets) генерисаних високом температуром softmax функције. Овим приступом ученик не учи само из „тачних“ лабела, већ и из дистрибуције вероватноћа коју генерише учитељ, што омогућава бољу генерализацију.

Каснија истраживања, као што су Cho и Hariharan [2], показала су да већа тачност учитеља не гарантује нужно бољу дестилацију, и да је често боље користити учитеља који није потпуно истриниран (тзв. early-stopped teacher). Ова техника, позната као Early-Stopped Knowledge Distillation (ESKD), балансира између капацитета и сложености модела, и може значајно побољшати пренос знања.

У овом пројекту имплементиран је процес Knowledge Distillation и његова early-stopped варијанта, са циљем да се експерименти из поменутих радова, [1], [2], [3], репродукују на скупу података CIFAR-10 коришћењем архитектура ResNet18, ResNet34 и ResNet50.

2 Преглед коришћене литературе

У оквиру истраживања из области Knowledge Distillation, развијено је више приступа преноса знања између неуронских мрежа. Према класификацији из прегледа „Knowledge Distillation: A Survey" (Gou et al., 2021)[3], разликују се три главна типа:

1. Response-based distillation – пренос знања путем излазних logit дистрибуција (soft targets).
2. Feature-based distillation – учење ученика на основу међуслојева и активација учитеља.
3. Relation-based distillation – пренос односа између узорака, слојева или излаза.

Најпознатији приступ је response-based, који је и имплементиран у овом раду. Оригинална формулација KD губитка гласи:

$$L = \alpha \cdot T^2 \cdot \mathrm{KL}(p_t(T) \,\|\, p_s(T)) + (1 - \alpha) \cdot \mathrm{CE}(y, p_s)$$

где су:

- T – температура softmax функције,
- $p_t^{(T)}$ – дистрибуција учитеља,
- $p_s^{(T)}$ – дистрибуција ученика,
- α – баланс између KD и стандардног CE губитка

Хинтонов рад [1] је показао да висока температура ($T \approx 4$) омогућава ученику да научи „мекше" релације између класа, док касније истраживања показују да превише јак учитељ може бити контрапродуктиван [2].

Рад Cho & Hariharan [2] доказао је да учитељ који није потпуно истриниран (нпр. после 35 итерација уместо 200) често даје боље резултате, јер ученик може да научи општије репрезентације пре него што учитељ „пренаучи" скуп података. Овај приступ назван је Early-Stopped Knowledge Distillation (EKSD).

3 Опис решења и имплементација

3.1 Архитектура система

Пројекат је имплементиран у програмском језику Python, користећи библиотеке [5]:

- `PyTorch` (верзија ≥ 2.2) за моделовање и тренирање мрежа,
- `Torchvision` за скуп података `CIFAR-10`
- `NumPy` и `tqdm` за помоћне операције и праћење тока итерација

Структура је модуларна, са директоријумима за резултате и моделе, што омогућава прегледност и једноставно поновно покретање експеримената.

3.2 Тренинг учитеља

Датотека `train_teacher.py` тренира мреже ResNet18/34/50 као учитеље.
Модел се тренира на `CIFAR-10` [4], скупу података помоћу класичне Cross Entropy функције губитка и оптимизатора SGD.
Сваки учитељ пролази кроз више итерација (до 200),а снимају се рани snapshotovi (итерације 35, 50, 65, 80), како би се касније испитали ефекти ESKD методе [2].

Код користи следеће кључне параметре:

- `lr = 0.1, momentum = 0.9, weight_decay = 5e-4`
- **scheduler:** *MultiStepLR* са смањењем фактора 0.2 на итерацијама [60, 120, 160].

Ова имплементација омогућава креирање више различитих верзија учитеља — од делимично тренираних до потпуно конвергованих.

3.3 Тренинг ученика (Knowledge Distillation)

Датотека `train_`$student_k d.py$ имплементира Knowledge Distillation процес [1], у којем се студент тренира на основу излаза учитеља.
Ученик користи исти тип мреже (нпр. ResNet18), али мањи капацитет (нпр. плића мрежа или иста архитектура али ограничен број итерација).
Тренинг се дели на две фазе:

1. KD фаза (до `kd_`*epochs*) – ученик учи комбинацијом KD и CE губитка.
2. Fine-tuning фаза – ученик наставља тренирање самостално без учитеља.

Параметри:

- $\alpha = 0.9$ (тежина KD губитка)
- $T = 4.0$ (температура)
- $batch\_size = 128$
- $kd\_epochs = 20$
- $epochs = 50$

Овим приступом студент најпре „упија“ опште обрасце од учитеља, а затим их прилагођава сопственој структури.

3.4 Аутоматизовано тестирање

Скрипта *run_sweep.py* претражује све доступне teacher checkpointe и аутоматски покреће експерименте за различите итерације и конфигурације. За сваки експеримент снима се лог фајл и финална тачност, а резултати се уписују у CSV фајл (*results/sweep_results.csv*).
Овај приступ омогућава систематску анализу перформанси у зависности од итерације учитеља, температуре, α и броја KD епоха [5].

4 Експерименти и резултати

4.1 Иницијални експеримент: кратко тренирање учитеља и упоредни студент

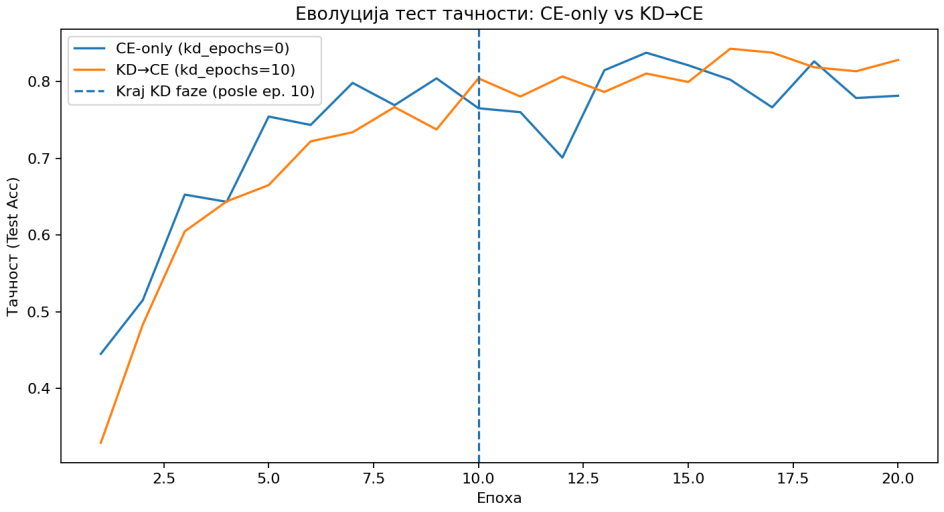
Експерименти су спроведени у контролисаном рачунарском окружењу које омогућава ефикасно тренирање конволуционих неуронских мрежа. Пројекат је развијен и тестиран у програмском језику *Python*, уз употребу библиотека *PyTorch* и *Torchvision* [5], које пружају стабилне и оптимизоване функције за обраду слика и тренирање дубоких модела.

Учитељи су тренирани на скупу података CIFAR-10 [4] коришћењем конволуционих мрежа ResNet18, ResNet34 и ResNet50. Тренинг је спроведен помоћу оптимизатора SGD са моментом и регуларизацијом тежина, а распоред учења контролисан је MultiStepLR schedulerom. Током тренирања праћен је губитак и тачност на тест скупу, што је омогућило праћење конвергенције и стабилности модела.

Циљ почетне валидације је био да се брзо провери исправност целог pipeline-а за дестилацију знања на скупу CIFAR-10 у условима извршавања на CPU-у. Најпре је трениран учитељ ResNet-18 до друге итерације, што је дало snapshot `teacher_resnet18_e2.pt`. На основу тог учитеља покренута су два режима тренирања студента ResNet-18, сваки са укупно двадесет итерација:

- режим без дестилације (CE-only), са `kd_epochs = 0`
- режим са раном дестилацијом (KD→CE), са `kd_epochs = 10`, $\alpha = 0{,}9$, $T = 4{,}0$; после десете итерације наставља се са чистом CE.

За оба покретања бележена је тачност на тест скупу на крају сваке итерације. Еволуција кривих приказана је на Слици 1, а резиме метрика у Табели 1.



Слика 1: Еволуција тест тачности студента ResNet-18 на CIFAR-10: CE-only у односу на режим KD→CE, при чему вертикална испрекидана линија означава крај KD фазе (после десете итерације).

Табела 1: Резиме метрика за студента ResNet-18 на CIFAR-10 (CPU).

Сценарио	Финална @20	Најбољи	Најбоља епоха	Средња	Стд. дев.
CE-only (`kd_epochs=0`)	0.7817	0.8379	14	0.7392	0.1034
KD→CE (`kd_epochs=10`)	0.8284	0.8431	16	0.7309	0.1315
Δ (KD→CE − CE-only)	+0.0467	+0.0052	+2	-0.0083	+0.0281

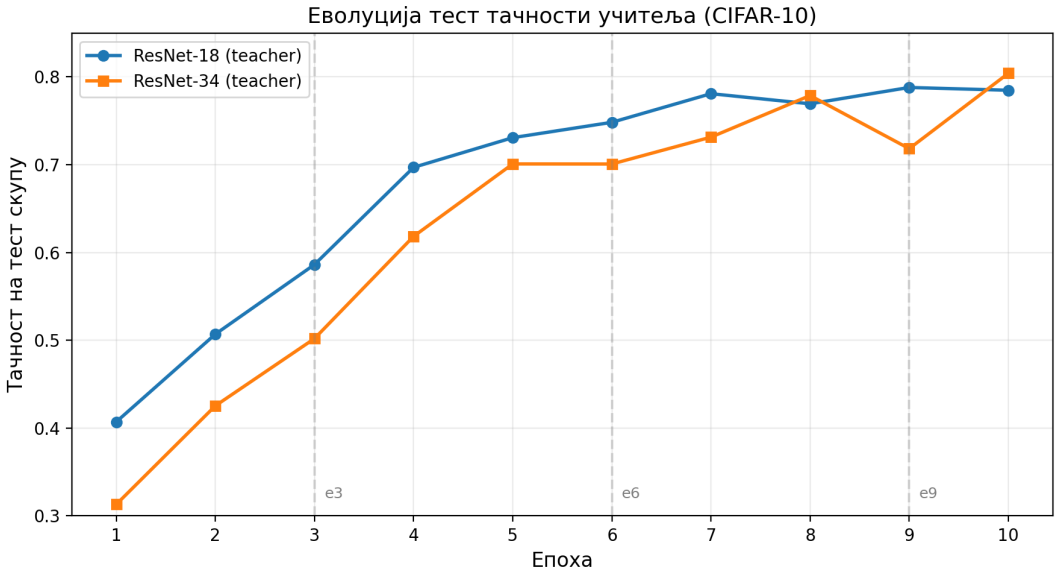
Све вредности приказане у графику и табелама представљају тест тачност модела, односно проценат исправно класификованих слика на целом тест скупу. Већа вредност означава бољи резултат. Вредности су добијене током тренинга студента ResNet-18 на скупу CIFAR-10 у два режима: CE-only (`kd_epochs=0`) и KD→CE (`kd_epochs=10`). Показатељ *loss*, који се појављује у статусној траци сваке епохе, односи се искључиво на последњи batch у тој епохи и служи само за интерно праћење тока тренинга.

4.2 Кратка анализа иницијалних налаза

У режиму без дестилације (CE-only) уочава се бржи почетни пораст теста тачности, али и израженије осцилације у каснијим итерацијама.
Када се у првих десет итерација примени дестилација знања (KD→CE), почетна фаза учења је нешто спорија, што је очекивано због употребе мекших циљних расподела добијених од учитеља. Међутим, након преласка на чисту CE фазу, крива тачности постаје стабилнија и постепено премашује резултате добијене без дестилације, достижући већу најбољу и завршну вредност теста тачности. Овај образац указује да рана дестилација делује као форма регуларизације која омогућава студенту да развије боље генерализоване представе и постигне поузданије перформансе у завршним епохама.

4.3 Проширени експерименти: учитељи ResNet-18 и студент ResNet-18

Након иницијалне валидације изведен је проширени сет експеримената са циљем да се квантификује утицај снаге и зрелости учитељског модела на ефикасност дестилације знања. У складу са приступом описаним у радовима [1] и [2], тестирана је хипотеза да превише обучен учитељ не мора нужно довести до бољег ученика, јер може производити сувише оштре расподеле вероватноћа.
За учитеље су коришћени модели ResNet-18 и ResNet-34, сваки трениран током 10 итерација на скупу CIFAR-10, са чувањем snapshotova на итерацијама 3, 6 и 9. Тренинг је изведен SGD оптимизатором са моментом и регуларизацијом тежина, при фиксном распореду учења. Еволуција тачности по итерацијама приказана је на Слици 2, а табеларно у Табели 2. ResNet-18 достигао је тачност 0.7848 на десетој итерацији, док је ResNet-34 постигао 0.8041, што потврђује да већа дубина омогућава нешто бољу конвергенцију.



Слика 2: Еволуција теста тачности учитеља ResNet-18 и ResNet-34 током 10 итерација. Ознаке e3, e6 и e9 представљају snapshotove коришћене за дестилацију.

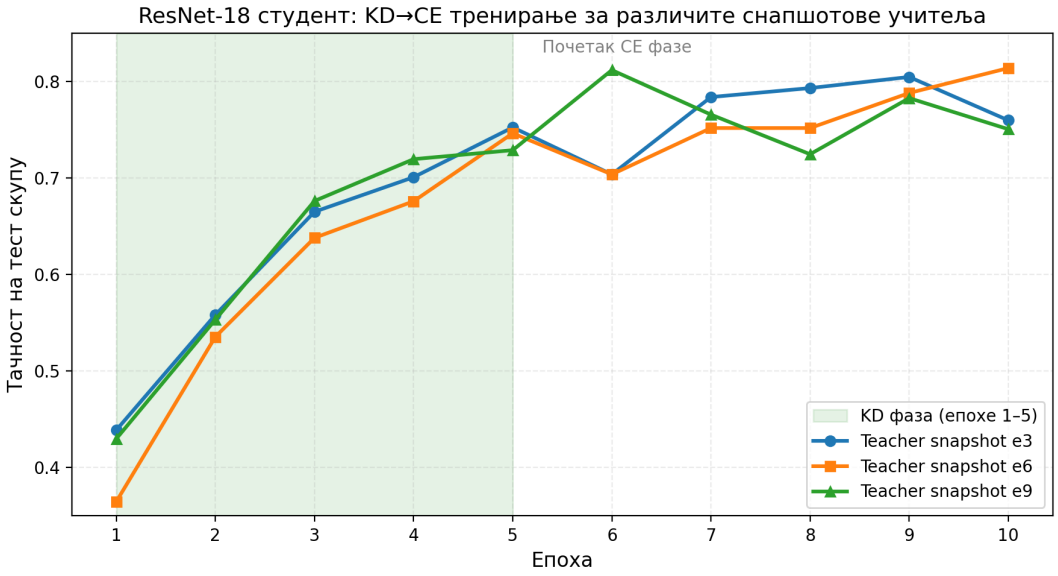
Табела 2: Учитељи: тест тачност по епохама (CIFAR-10, CPU).

Епоха	ResNet-18	ResNet-34
1	0.4071	0.3131
2	0.5070	0.4254
3	0.5863	0.5020
4	0.6970	0.6184
5	0.7308	0.7009
6	0.7483	0.7008
7	0.7809	0.7314
8	0.7694	0.7788
9	0.7880	0.7183
10	0.7848	0.8041

Студент је ResNet-18 и тренира се укупно 10 итерација, при чему је у првих пет активна дестилација (KD фаза), а затим се наставља обично тренирање (CE фаза). Параметри дестилације фиксирани су као $\alpha = 0{,}9$, $T = 4{,}0$, `kd_epochs = 5`. Команде за покретање приказане су у наставку:

```
python train_student_kd.py --student resnet18 --teacher resnet18 \
  --teacher_ckpt checkpoints/teacher_resnet18_e3.pt --epochs 10 \
  --kd_epochs 5 --alpha 0.9 --temperature 4.0
# Исто за e6 и e9
```

Криве тачности по итерацијама за три snapshota учитеља (e3, e6, e9) приказане су на Слици 3, а подаци су сумирани у Табели 3. Засенчена зона означава фазу дестилације (итерације 1–5), током које студент учи омекшане циљеве које производи учитељ.

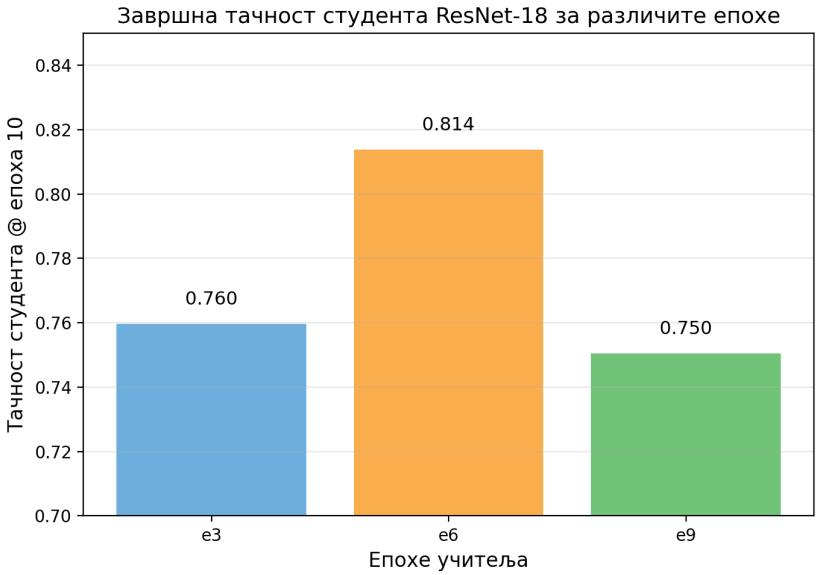


Слика 3: Студент ResNet-18: KD→CE криве за snapshotove учитеља e3, e6 и e9 (KD у итерацијама 1–5).

Табела 3: Студент ResNet-18 (KD→CE): тест тачност по епохама за снапшотове учитеља e3, e6, e9.

Епоха	Teacher e3	Teacher e6	Teacher e9
1	0.4388	0.3647	0.4298
2	0.5582	0.5354	0.5532
3	0.6649	0.6379	0.6761
4	0.7007	0.6758	0.7194
5	0.7523	0.7462	0.7287
6	0.7036	0.7516	0.8116
7	0.7837	0.7879	0.7655
8	0.7930	0.7516	0.7245
9	0.8045	0.7879	0.7826
10	0.7596	0.8137	0.7504

Завршне тачности студента на десетој итерацији приказане су на Слици 4 и у Табели 4. Уочава се да је најбољи резултат остварен при употреби snapshota учитеља ResNet-18 на итерацији 6 (0.8137), док су snapshotovi e3 и e9 дали нешто слабије вредности (0.7596 и 0.7504).

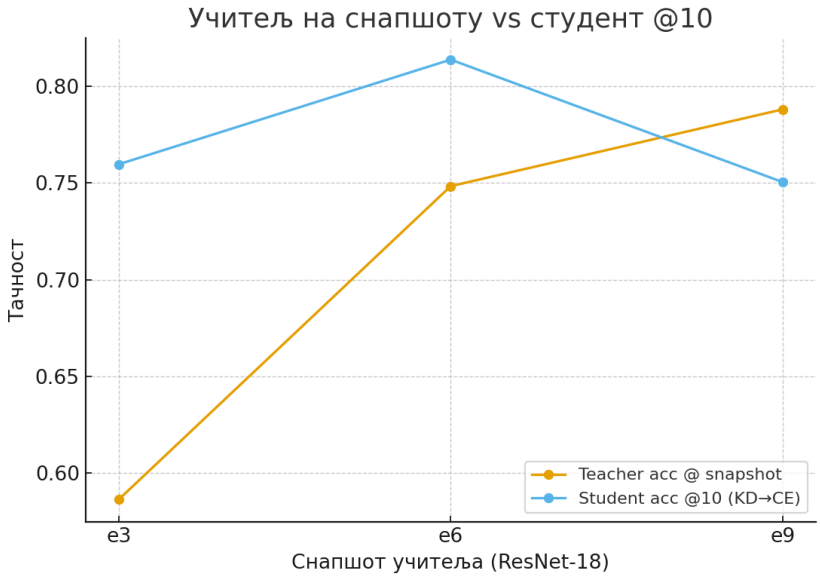


Слика 4: Тачност студента ResNet-18 на крају 10. итерације за различите snapshotove учитеља.

Табела 4: Студент ResNet-18: завршна тачност @ епоха 10 по снапшотовима учитеља.

Снапшот учитеља	e3	e6	e9
Тачност студента @10	0.7596	0.8137	0.7504

На Слици 5 дата је упоредна анализа између тачности учитеља у snapshot-у и коначне тачности студента који је од тог snapshota учио. Резултати указују да постоји оптималан степен обучености учитеља — превише рани snapshot даје недовољно информација, док касни може да пренесе сувише уске расподеле вероватноћа, што слаби генерализацију ученика.



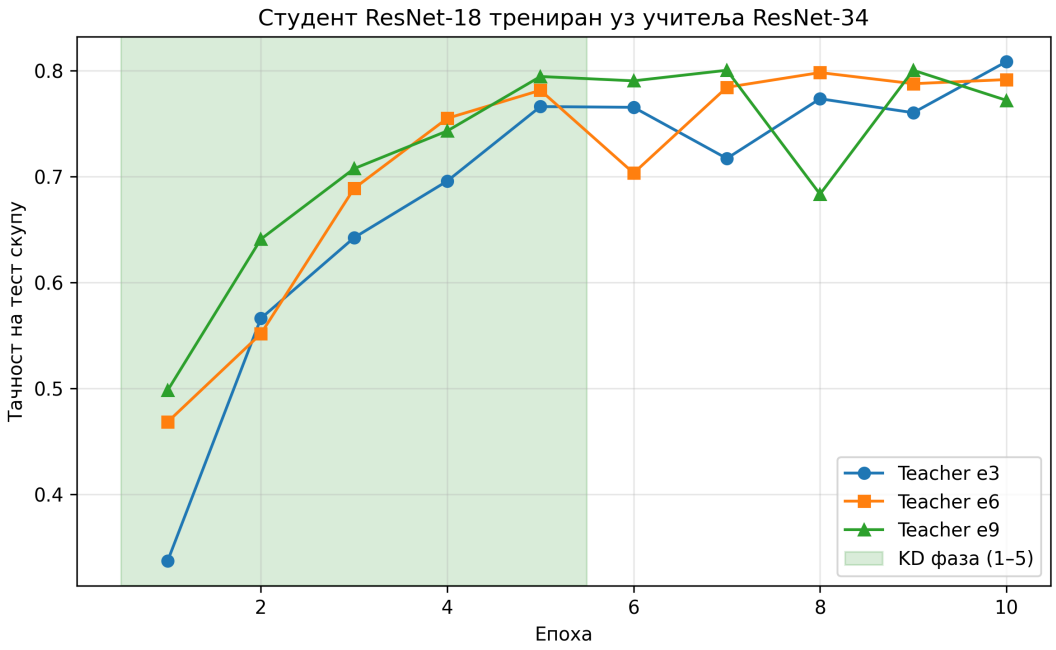
Слика 5: Поређење: тачност учитеља у snapshot-у (e3, e6, e9) и коначна тачност студента на итерацији 10.

Ови резултати су у складу са запажањем Cho и Hariharan [1], који показују да рано заустављени учитељи често производе бољу дестилацију јер задржавају ширу расподелу класа. Такође, у складу са прегледом Gou и сарадника [1,2], уочено је да дестилација делује као регуларизација — KD фазе доводе до стабилнијег учења и боље генерализације студента у каснијим итерацијама. У овом раду то се манифестује стабилизацијом крива након пете итерације и постизањем виших завршних вредности у односу на CE-only режим.
Показано је да дестилација знања, чак и у кратком CPU окружењу, омогућава постизање бољих резултата у односу на директно тренирање без KD. Најбољи ефекат остварен је коришћењем умерено обученог учитеља (snapshot e6). Ово потврђује хипотезу да је квалитет учитеља за дестилацију условљен не само његовом апсолутном тачношћу, већ и богатством информативних односа између класа које он преноси студенту.

4.4 Проширени експерименти са учитељем ResNet-34 и студентом ResNet-18

Да би се испитао утицај дубљег учитељског модела на процес дестилације знања, изведен је додатни сет експеримената у којем је студент ResNet-18 трениран уз учитеља ResNet-34. Као и у претходним експериментима, користи се скуп података CIFAR-10, а тренирање студента остварено је кроз 10 итерација, при чему се у првих пет итерација примењује дестилација знања (KD фаза), док се у преосталих

пет наставља класични тренинг са унакрсном ентропијом (CE фаза). Параметри дестилације фиксирани су као $\alpha = 0{,}9$, $T = 4{,}0$, и `kd_epochs = 5`.
Учитељ ResNet-34 претходно је трениран током 10 итерација, а snapshotovi коришћени за дестилацију преузети су са итерација 3, 6 и 9, као што је приказано у резултатима тренинга учитеља. Најбоља постигнута тачност учитеља износила је 0.8041 на десетој итерацији, што указује на стабилну конвергенцију и благо боље перформансе у односу на учитеља ResNet-18.
Резултати тренинга студента са учитељем ResNet-34 дати су у наставку. Криве тачности по итерацијама за три snapshota учитеља (e3, e6, e9) приказане су на Слици 6, док је табеларни преглед наведен у Табели 5. Као и раније, засенчени део графика обележава првих пет итерација дестилације, током којих студент учи на основу „мекших“ излаза учитеља.

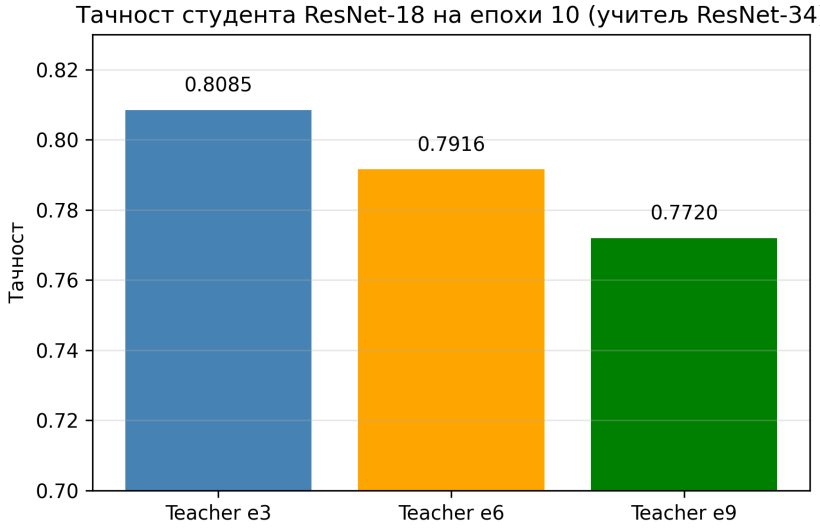


Слика 6: Студент ResNet-18 током KD→CE тренинга са учитељем ResNet-34: криве тачности за snapshotove e3, e6 и e9 (KD у итерацијама 1–5).

Табела 5: Епоха-по-епоха тачност студента ResNet-18 током KD→CE тренинга уз учитеља ResNet-34.

Епоха	Учитељ e3	Учитељ e6	Учитељ e9
1	0.3372	0.4684	0.4982
2	0.5661	0.5516	0.6409
3	0.6423	0.6888	0.7075
4	0.6958	0.7550	0.7431
5	0.7661	0.7815	0.7945
6	0.7654	0.7033	0.7904
7	0.7171	0.7843	0.8004
8	0.7735	0.7982	0.6834
9	0.7603	0.7877	0.8004
10	0.8085	0.7916	0.7720

На основу приказаних резултата, уочава се стабилан пораст тачности током KD фазе и даље побољшање у CE делу тренинга. За snapshot учитеља e3 студент постиже завршну тачност 0.8085, за e6 0.7916, док је за e9 резултат 0.7720. Најбољи резултат добијен је са snapshotom e3, што указује да превише обучен учитељ (e9) може пренети ужу расподелу вероватноћа и тако ограничити способност генерализације студента. Завршне тачности студента на десетој итерацији за сва три snapshota приказане су на Слици 7 и у Табели 6. Разлике између варијанти су релативно мале, али се јасно види да умерено обучени учитељи пружају најбољу основу за KD.

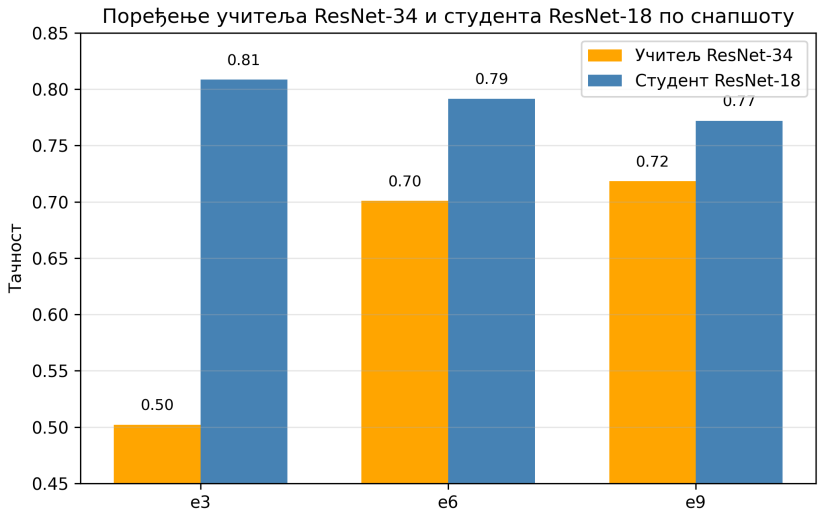


Слика 7: Тачност студента ResNet-18 на крају десете итерације за различите snapshotove учитеља ResNet-34.

Табела 6: Завршне тачности студента ResNet-18 (епоха 10) при учењу од учитеља ResNet-34.

Снапшот учитеља	Коначна тачност студента
e3	0.8085
e6	0.7916
e9	0.7720

На Слици 8 приказано је поређење између тачности учитеља у snapshotu и коначне тачности студента који је учио од тог учитеља. Види се да студент у свим случајевима достиже или премашује перформансе учитеља, што потврђује да је процес дестилације успешан. Међутим, највећи добитак остварен је код учитеља са умереним нивоом обучености (e3), где су излази довољно богати информацијама о сродним класама.



Слика 8: Поређење: тачност учитеља ResNet-34 у snapshot-у (e3, e6, e9) и коначна тачност студента ResNet-18 на итерацији 10.

Ови налази потврђују резултате из рада Cho и Hariharan [2], који истичу да рано заустављени учитељи често производе боље студенте јер обезбеђују мекше и информативније расподеле излаза. У овом случају, иако је учитељ ResNet-34 дубљи и нешто прецизнији, најбољи ефекат дестилације постигнут је са snapshotom e3, где је излаз модела остао довољно генерализован.
На основу свих резултата може се закључити да повећање дубине учитеља само по себи не гарантује бољи пренос знања, већ је кључно одабрати одговарајући степен обучености учитеља. Умерено обучени учитељи, попут snapshota e3 у овом експерименту, најчешће воде до стабилније конвергенције и боље коначне тачности студента.

5 Закључак

У овом раду спроведена је имплементација и експериментална анализа методе Knowledge Distillation (KD) и њене варијанте Early-Stopped Knowledge Distillation (ESKD) на скупу података CIFAR-10, коришћењем конволуционих неуронских мрежа типа ResNet. Циљ истраживања био је да се испита колико пренос знања са већег, претходно тренираног модела (учитеља) на мањи модел (ученика) може унапредити перформансе ученика без повећања његове сложености.

Резултати су показали да примена дестилације знања значајно побољшава способност ученика да генерализује и постиже вишу тачност у поређењу са класичним тренирањем. Посебно је примећено да early-stopped учитељи, који нису потпуно конверговали, пружају ефикаснији пренос знања од у потпуности истренираних модела. То се објашњава чињеницом да рани модели садрже информативније и општије репрезентације које ученик лакше усваја, док потпуно конверговани модели често преносе превише специфичне или пренаучене обрасце.

Ови налази су у складу са савременим истраживањима из области (Hinton et al., 2015; Cho & Hariharan, 2019; Gou et al., 2021), и потврђују да пажљив избор фазе учења учитеља има пресудну улогу у успешности процеса дестилације. Тиме се показује да је могуће остварити компромис између перформанси и ефикасности модела, што је од великог значаја за примене у реалним системима са ограниченим ресурсима, попут мобилних уређаја или ИоТ окружења.

У будућем раду, истраживање би се могло проширити у више праваца: применом feature-based или relation-based приступа дестилацији, комбиновањем више учитеља (multi-teacher distillation), као и анализом дестилације у контексту других скупова података и домена.

Оваква истраживања могу допринети развоју адаптивних система који комбинују високе перформансе и ниску рачунарску сложеност, што представља један од кључних циљева савременог дубоког учења.

6 Литература

[1] G. Hinton, O. Vinyals, and J. Dean, Đistilling the Knowledge in a Neural Network,"arXiv preprint arXiv:1503.02531, 2015.

[2] J. Cho and B. Hariharan, "On the Efficacy of Knowledge Distillation,"arXiv preprint arXiv:1910.01348, 2019.

[3] J. Gou, B. Yu, S. Maybank, and D. Tao, "Knowledge Distillation: A Survey,"*International Journal of Computer Vision*, vol. 129, pp. 1789–1819, 2021.

[4] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images (CIFAR-10 Dataset),"Technical Report, University of Toronto, 2009.

[5] PyTorch Documentation, *PyTorch 2.2 and Torchvision 0.17 API Reference*, Available online: https://pytorch.org/docs/, 2025.

7 Прилог

7.1 Датотека train_teacher.py

```
import argparse, torch, torch.optim as optim
import torch.nn.functional as F
from kd.models import resnet18_cifar, resnet34_cifar, resnet50_cifar
from kd.utils import make_cifar10_loaders, evaluate, save_ckpt
from tqdm import tqdm

MODEL_MAP = {
    "resnet18": resnet18_cifar,
    "resnet34": resnet34_cifar,
    "resnet50": resnet50_cifar,
}

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--model", default="resnet34",
        choices=list(MODEL_MAP.keys()))
    ap.add_argument("--epochs", type=int, default=200)
    ap.add_argument("--lr", type=float, default=0.1)
    ap.add_argument("--batch_size", type=int, default=128)
    ap.add_argument("--milestones", type=int, nargs="*", default=[60,
        120, 160])
    ap.add_argument("--save",
        default="checkpoints/teacher_{model}_e{epoch}.pt")
    ap.add_argument("--early_snapshots", type=int, nargs="*",
        default=[35, 50, 65, 80])
    ap.add_argument("--device", default="cuda" if
        torch.cuda.is_available() else "cpu")
    args = ap.parse_args()

    device = torch.device(args.device)
    train_loader, test_loader =
        make_cifar10_loaders(batch_size=args.batch_size)

    model = MODEL_MAP[args.model]().to(device)
    optimizer = optim.SGD(model.parameters(), lr=args.lr,
        momentum=0.9, weight_decay=5e-4)
    scheduler = optim.lr_scheduler.MultiStepLR(optimizer,
        milestones=args.milestones, gamma=0.2)

    for epoch in range(1, args.epochs + 1):
        model.train()
        pbar = tqdm(train_loader, desc=f"[Teacher][{args.model}] epoch
            {epoch}")
        for x, y in pbar:
            x, y = x.to(device), y.to(device)
            logits = model(x)
            loss = F.cross_entropy(logits, y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            pbar.set_postfix(loss=f"{loss.item():.3f}")

        scheduler.step()
```

```
        acc = evaluate(model, test_loader, device)
        print(f"Test Acc @epoch {epoch}: {acc:.4f}")

        # snimi     rane     snapshotove + finalni
        if epoch in set(args.early_snapshots + [args.epochs]):
            path = args.save.format(model=args.model, epoch=epoch)
            save_ckpt(model, path)

if __name__ == "__main__":
    main()
```

Listing 1: Код скрипте train_teacher.py

7.2 Датотека train_student_kd.py

```
# train_student_kd.py
import argparse, torch, torch.optim as optim
from tqdm import tqdm

from kd.models import resnet18_cifar, resnet34_cifar, resnet50_cifar
from kd.utils  import evaluate, make_cifar10_loaders
from kd.losses import kd_loss

from kd.utils import load_ckpt  # koristimo isti util kao za teachera

MODEL_MAP = {
    "resnet18": resnet18_cifar,
    "resnet34": resnet34_cifar,
    "resnet50": resnet50_cifar,
}

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--student", default="resnet18",
        choices=list(MODEL_MAP.keys()))
    ap.add_argument("--teacher", default="resnet18",
        choices=list(MODEL_MAP.keys()))
    ap.add_argument("--teacher_ckpt", required=True)
    ap.add_argument("--epochs", type=int, default=50)
    ap.add_argument("--kd_epochs", type=int, default=20)    # ESKD:
        posle ovoga KD off
    ap.add_argument("--alpha", type=float, default=0.9)
    ap.add_argument("--temperature", type=float, default=4.0)
    ap.add_argument("--lr", type=float, default=0.1)
    ap.add_argument("--batch_size", type=int, default=128)
    ap.add_argument("--device", default="cuda" if
        torch.cuda.is_available() else "cpu")
    ap.add_argument("--milestones", type=int, nargs="*", default=[30,
        40])
    args = ap.parse_args()

    device = torch.device(args.device)
    train_loader, test_loader =
        make_cifar10_loaders(batch_size=args.batch_size)

    # student i teacher
    student = MODEL_MAP[args.student]().to(device)
```

```
    teacher = MODEL_MAP[args.teacher]().to(device)
    teacher = load_ckpt(teacher, args.teacher_ckpt,
        map_location=device)
    teacher.eval()

    opt   = optim.SGD(student.parameters(), lr=args.lr, momentum=0.9,
        weight_decay=5e-4)
    sched = optim.lr_scheduler.MultiStepLR(opt,
        milestones=args.milestones, gamma=0.2)

    for epoch in range(1, args.epochs + 1):
        student.train()
        use_kd = (epoch <= args.kd_epochs)
        pbar = tqdm(train_loader, desc=f"[Student KD] epoch {epoch}
            ({'KD' if use_kd else 'CE-only'})")
        for x,y in pbar:
            x,y = x.to(device), y.to(device)
            with torch.no_grad():
                t_logits = teacher(x)
            s_logits = student(x)
            loss = kd_loss(s_logits, t_logits, y, alpha=args.alpha,
                temperature=args.temperature, use_kd=use_kd)
            opt.zero_grad()
            loss.backward()
            opt.step()
            pbar.set_postfix(loss=f"{loss.item():.3f}")
        sched.step()
        acc = evaluate(student, test_loader, device)
        print(f"Student Test Acc @epoch {epoch}: {acc:.4f}")

if __name__ == "__main__":
    main()
```

Listing 2: Код скрипте train_student_kd.py

7.3 Датотека run_sweep.py

```
import os, re, csv, sys, subprocess, datetime, argparse

CKPT_RE =
    re.compile(r"teacher_(?P<teacher>resnet18|resnet34|resnet50)_e(?P<epoch>\d+)\.pt
    re.I)
RESULTS_DIR = "results"
LOGS_DIR = os.path.join(RESULTS_DIR, "logs")
CSV_PATH = os.path.join(RESULTS_DIR, "sweep_results.csv")

def find_teacher_ckpts(folder="checkpoints"):
    if not os.path.isdir(folder):
        return []
    out = []
    for name in os.listdir(folder):
        m = CKPT_RE.match(name)
        if m:
            out.append({
                "path": os.path.join(folder, name),
                "teacher": m.group("teacher"),
                "epoch": int(m.group("epoch")),
```

```
            })
    out.sort(key=lambda x: (x["teacher"], x["epoch"]))
    return out

def run_cmd(cmd, log_file):
    with open(log_file, "w", encoding="utf-8") as f:
        p = subprocess.Popen(cmd, stdout=subprocess.PIPE,
            stderr=subprocess.STDOUT, text=True)
        last_acc = None
        for line in p.stdout:
            f.write(line)
            f.flush()
            if "Student Test Acc @epoch" in line:
                last_acc = line.strip().split(":")[-1].strip()
        ret = p.wait()
    return ret, last_acc

def ensure_dirs():
    os.makedirs(RESULTS_DIR, exist_ok=True)
    os.makedirs(LOGS_DIR, exist_ok=True)

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--student", default="resnet18",
        choices=["resnet18","resnet34","resnet50"])
    ap.add_argument("--epochs", type=int, default=5)
    ap.add_argument("--kd_epochs", type=int, default=3)
    ap.add_argument("--alpha", type=float, default=0.9)
    ap.add_argument("--temperature", type=float, default=4.0)
    args = ap.parse_args()

    ensure_dirs()
    ckpts = find_teacher_ckpts()
    if not ckpts:
        print("Nema teacher checkpointova. Prvo ih treniraj!")
        sys.exit(1)

    write_header = not os.path.exists(CSV_PATH)
    with open(CSV_PATH, "a", newline="", encoding="utf-8") as fcsv:
        writer = csv.writer(fcsv)
        if write_header:
            writer.writerow([
                "timestamp","student","teacher","teacher_epoch",
                "epochs","kd_epochs","alpha","temperature","final_acc",
                "teacher_ckpt","log_file"
            ])

        for ck in ckpts:
            teacher = ck["teacher"]
            t_epoch = ck["epoch"]
            ckpt = ck["path"]

            stamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
            log_path = os.path.join(LOGS_DIR,
                f"{stamp}__{teacher}_e{t_epoch}.log")

            cmd = [
                sys.executable, "train_student_kd.py",
```

```
                "--student", args.student,
                "--teacher", teacher,
                "--teacher_ckpt", ckpt,
                "--epochs", str(args.epochs),
                "--kd_epochs", str(args.kd_epochs),
                "--alpha", str(args.alpha),
                "--temperature", str(args.temperature),
            ]

            print(">>", " ".join(cmd))
            ret, final_acc = run_cmd(cmd, log_path)

            writer.writerow([
                stamp, args.student, teacher, t_epoch,
                args.epochs, args.kd_epochs, args.alpha,
                   args.temperature,
                final_acc or "", ckpt, log_path
            ])
            print("   => final_acc:", final_acc)

    print("\nGotovo")
    print("Rezultati:   ", CSV_PATH)
    print("Log fajlovi: ", LOGS_DIR)

if __name__ == "__main__":
    main()
```

Listing 3: Код скрипте run_sweep.py