# Универзитет Унион

## Рачунарски факултет



# Класификација пнеумонијалних налаза са конволутивним неуронским мрежама

Студент: Петар Арнаутоивић РМ 74/16 Београд, 2021

Ментор: проф. Др Јелена Васиљевић

# САДРЖАЈ

[Увод 3](#_Toc101110401)

[Упала плућа – пнеумонија 4](#_Toc101110402)

[Дубоко учење 5](#_Toc101110403)

[Разлика између дубоког и машинског учења 5](#_Toc101110404)

[Неуронске мреже 7](#_Toc101110405)

[Неурон 8](#_Toc101110406)

[Потпуно повезане неуронске мреже 10](#_Toc101110407)

[Конволутивне неуронске мреже 13](#_Toc101110408)

[Примењене методе у овом раду 19](#_Toc101110409)

[Прикупљање података 19](#_Toc101110410)

[Припрема података 22](#_Toc101110411)

[Дефинисање модела 24](#_Toc101110412)

[Креирање модела 25](#_Toc101110413)

[Тренирање модела 29](#_Toc101110414)

[Пренесено учење 31](#_Toc101110415)

[Прецизно подешавање 35](#_Toc101110416)

[Поређење перформанси 37](#_Toc101110417)

[Закључак 41](#_Toc101110418)

[Референце 42](#_Toc101110419)

# Увод

Одувек је човек покушавао да разуме шта је то интелигенција и како она фукнционише, како ми заправо размишљамо и учимо. На основу којих параметара доноси закључке и одлуке, предвиђа и тумачи одређене проблеме. То су нека питања на које и данас човечанство не може у потпуности да одговори.

Вештачка интелигенција (Artificial Intelligence) покушава не само да одговори на та питања и да их разуме, већ да опонаша људску интелигенцију. Данас, вештачка интелигенција је напредна област са много практичних делова и изазовних истраживачких тема. Интелигентни софтвери, попут препознавање говора, алгоритама за социјалне мреже, паметни асистенти (Alexa, Google Home, Apple HomePod), све су више у дневном животу присутни.

Вештачку интелигенцију први је предложио Џон Макарти 1956. Идеја, коју је предложио математичар Алан Тјуринг, била је да се направе машине које имају способност да мисли и учи као човек. Алан Тјуринг је своје идеје и хипотезе спровео у праксу, тако што је почео да тестира да ли машине могу да мисле. Након серија тестирања, метода је добила назив Тјурингов тест. Испоставило се да машине имају способност да уче слично као људи. Тјурингов тест се користи као прагматичан начин тестирања, како би се могло оценити да ли машина може да размишља као човек.

У овом раду покушаћу да објасним неке од најважнијих делова вештачке интелигенције, њене предности и мане, као и саму примену вештачке интелигенције у разним пољима науке и рада, највише о доприносу вештачке интелигенције у медицини. Описаћу проблем класификације рендгенских налаза плућа са пнеумонијом и обучити програм да сам може да класификује на основу претходних тренинга. Користићу методе дубоког учења, као што су конволуционе неуронске мреже. Користићу три различита приступа за класификацију слика: класичну конволуциону неуронску мрежу (Convolutional Neural Network), Пренесено учење (Transfer Learning), Прецизно подешавање (Fine Tuning).

# Упала плућа – пнеумонија

Пнеумонија је инфекција у једном или оба плућа узрокована бактеријама, вирусима или гљивицама. Инфекција доводи до упале у ваздушним кесама плућа, које се називају алвеоли. Алвеоле се пуне течношћу или гнојем, што отежава дисање. И вирусна и бактеријска пнеумонија су заразне. То значи да се могу ширити са особе на особу удисањем капљица у ваздуху од кашља. Упала плућа представља најопасније обољење респираторног система и може бити узрокована различитим микроорганизмима. Типичне упале плућа изазване су бактеријама.

Такође можете добити ове врсте упале плућа ако дођете у контакт са површинама или предметима који су контаминирани бактеријама или вирусима који изазивају упалу плућа.

До обољења долази када су одбрамбени механизми респираторног система поремећени или одбрамбене снаге организма смањене. Као предиспонирајући чиниоци често се наводе расхлађење организма, употреба алкохола, примена анестезије, инфекције горњих дисајних путева и хронични бронхитис.

Упала плућа може бити у распону озбиљности од благе до опасне по живот. Најозбиљније је за одојчад и малу децу, особе старије од 65 година и особе са здравственим проблемима или ослабљеним имунолошким системом.

Упала плућа је и данас чест узрок смртности, упркос савременој антибиотској терапији. Пнеумонија се односи 10 пута више живота од свих осталих заразних болести. У земљама у развоју са ниским стандардом и лошим условима живота инфекције доњих дисајних путева најчешћи су узрок умирања.

# Дубоко учење

Дубоко  учење (Deep learning) је техника обучавања рачунара  примером која је део машинског учења (Machine learning). Дубоко учење добија све већи значај у последње време. Постизањем невероватних резултата који раније нису били могући, омогућили су развој софтвера као што су: гласовна контрола, аутомобили са аутопилотом, роботика, препознавање говора, итд.

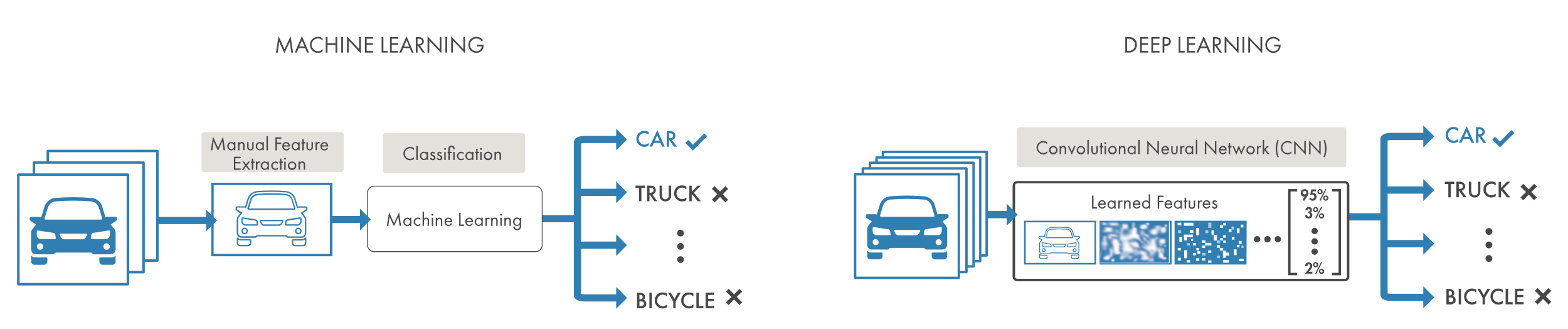
У дубоком учењу, модел учи да решава задатке као што су класификација помоћу слика, звука или текста. Такви модели, уколико су добро утренирани, могу да постигну већу тачност него човек. Да би тачност била што већа, модел мора бити добро утрениран, а то је могуће уколико користи велики скуп података и вишеслојну архитектуру неуронских мрежа.

Термин „дубоко“ се односи на већи број скривених слојева у архитектури неуронских мрежа. Основне неуронске мрежа садже два до три скривена слоја, док комплексиније архитектуре имају  знатно већи број.

## Разлика између дубоког и машинског учења

Дубоко учење је посебан облик машинског учења. Машинско учење (Machine Learning) је проучавање компјутерских алгоритама који се могу аутоматски побољшати кроз искуство и коришћење података. Процес рада машинског учења почиње тако што се релевантне функције ручно издвајају из података. Карактеристике се затим користе за креирање модела који категорише објекте на слици. У процесу рада дубоког учења, релевантне функције се аутоматски издвајају из слика. Поред тога, дубоко учење користи „end-to-end learning” (учење од краја до краја), што значи да се мрежи дају необрађени подаци и задатак који треба да изврши, уз то и учи како да се изврше аутоматски. Друга велика разлика је да се алгоритми дубоког учења скалирају са подацима, док код машинског, учење конвергира.

Када додате више примера и података у обуку, перформансе учења се заустављају на одређеном нивоу, док код дубоког учења није такав случај. Кључна предност дубоког учења је у томе што се перформансе побољшавају са повећањем броја података.



Илустрација 1:Поређење приступа машинског учења са дубоким учењем

У машинском учењу, ручно бирате карактеристике и класификатор за сортирање података. Са дубоким учењем, екстракција карактеристика и кораци моделирања су аутоматски. Maшинско учење нуди низ метода и модела које се могу ускладити на основу апликације, величине података које обрађује и врсте проблема које треба да се реши. Апликација, која користи дубоко учење захтева веома велику количину података (хиљаде слика) да би успешно обучио модел, као и графичку картицу (GPU) или графичке процесорске јединице, за ефикасну обраду података.

Уколико треба да направите избор између машинског учења и дубоког учења, размислите да ли графичка картица коју имате има високе перформансе. Ако перформансе вашег рачунара нису на одговарајућем нивоу, машинско учење би био бољи избор уместо дубоког учења. Дубоко учење је захтевније и генерално је сложеније, тако да је потребно најмање неколико хиљада слика како би добили жељене резултате. Поседовање графичке картице високих перформанси значи да ће моделу требати мање времена да анализира податке.

Diagram

Description automatically generated

Илустрација 2:Приказ односа дубоког и машинског учења

# Неуронске мреже

Неуронске мреже престављају једну од најбољих и најуспешнијих метода у машинском учењу. Први пут су представљене педесетих година прошлог века, али кроз своју историју нису биле приоритет научницима. Значајни помаци дешавају се 2006. године са појавом дубоких неуронских мрежа, а затим прави успон добијају 2012. године када дубоке неуронске мреже значајно надмашују претходне методе у решавању различитих рачунарских проблема. У неким задацима, превазишле су могућности и људских експерата. Данас представљају један од најбољих метода машинског учења са најширим доменом примене.Постоје различите врсте неуронских мрежа, развијене за специјализоване примене. Свака мрежа има одређен број елемената које називамо неуронима, по томе су препознатљиве све неуронске мреже. Као у људској анатомији, неурони прослеђују једни другима сигнале и израчунавају нове сигнале на основу претходно прослеђених сигнала. Начин повезаности неурона и на који начин они рачунају, одређују архитектуру мреже и прецизно одређују о каквој мрежи се ради. У зависности од врсте проблема, експерт покушава да дефинише најбоље одговарајућу архитектуру мреже. Да би најбоље објаснили принцип неуронских мрежа потребно је прво описати рад појединачног неурона.

# Неурон

Неурон је основна јединица израчунавања процеса и повезивањем више неурона добијамо неуронску мрежу. Рад неурона сличан је као код биолошког неурона, на улазу примају сигнале од других неурона и трансформишу добијени сигнал као свој излазни сигнал. Сви сигнали представљени су реалним бројевима, а обрада тих сигнала врши се неком математичком методом која укључује линеарну комбинацију улазних сигнала, а затим неалинеарну трасформацију те линеарне комбинације.

Дефиниција: „Неурон је функција *n* аргумената x1, x2, x3,…,xn која има општи облик:

где су 𝑤0, 𝑤1, . . . , 𝑤n параметри или тежине неурона, којима се врши линеарно комбиновање, а 𝑔 : R → R је нелинеарна активациона функција којом се та линеарна комбинација трансформише.“

Diagram

Description automatically generated

Илустрација 3:Неурон

Није одређено стриктно коју активациону фукцију треба користити, постоји више избора у зависности од архитектуре неуронске мреже. Активациона функција, у већини случајева, треба да буде монотона, непрекидна и диференцијабилна. Најчешће коришћене функције су: сигмоидна функција, хиперболички тангенс, исправљена линеарна јединица (rectified linear unit - ReLU), степенаста функција, Гаусова функција, лиенарна функција и њихове варијације. У свим слојевима мреже осим у последњим најчешће се користи нека модификација исправљене линеарне функције због успеш них резултата те функције.

Diagram

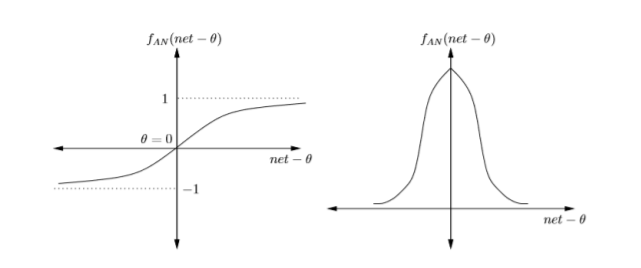
Description automatically generated

Илустрација 4: линеарна и степенаста функција

A picture containing text, antenna, day

Description automatically generated

Илустрација 5:Исправљена линеарна јединица (ReLU) и сигмоидна функција



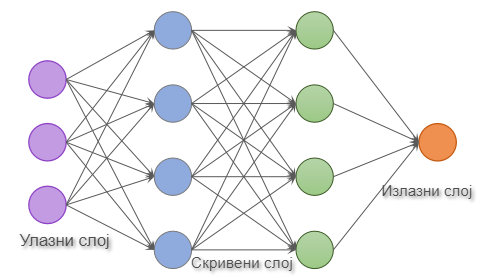
Илустрација 6: Хиперболички тангенс и Гаусова функција

Излазни сигнал је јачи, уколико је вредност линеарне комбинације већа, због моностоности активационе функције. Што су вредности улазних тежина веће, то је већи утицај тих улаза на излазни сигнал, а кад је ближа нули тај утицај се смањује.

Када се сигмоидна функција постави као активациона функција мреже, неурон постаје модел логистчке регресије, а када се постави идентититет за активациону функцију, неурон постаје модел линеарне регресије. Зато су неуронске мреже способне да извршавају класификацију и регресију.

# Потпуно повезане неуронске мреже

Потпуно повезане неуронске мреже (Fully connected neural networks) представљају традиционалан модел неуронских мрежа. То је најкоришћенији модел до сада. Као што из назива може да се претпостави, њихова структура је таква да се неурони слажу у слојеве, групе неурона, и повезане су тако да неурони једног слоја примају као улазе вредности свих неурона претхходног слоја, а вредности које се добију на том слоју прослеђују се неуронима следећег слоја.

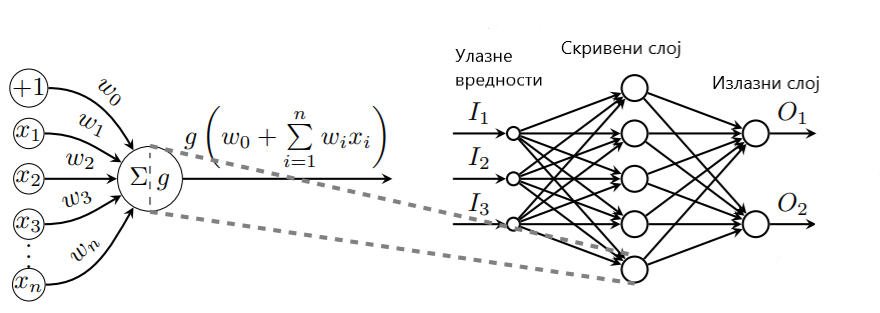


Илустрација 7: Структура потпуно повезане неуронске мреже

На слици 7 приказана је архитектура потпуно повезане неуронске мреже. На слици се могу уочити да се архитектура састоји из улазног слоја, скривених слојева и излазног слоја. Приказана су два скривена слоја неурона, који се називају тако јер се вредности тог слоја се не користе ван модела. Број скривених слојева није ограничен (1, … ,N) и због тога се називају дубоке неуронске мреже (deep neural networks). Већина мрежа које данас можемо да користимо су дубоке.

Дефиниција потпуно повезаних мрежа: „Потпуно повезана неуронска мрежа је функција ***𝑓w*** дефинисана на следећи начин:

где ***x*** представља улазни вектор, ***L*** је број слојева**, *hi***представља вектор вредности ***i-***тог слоја, ***Wi***  је матрица параметара чије врсте представљају векторе параметара неурона ***i-***тог слоја који множе улазне вредности, ***bi***  је вектор слободних коефицијената тих неурона, а ***g*** и ***g*** су активационе функције. Матрице ***Wi***  и вектори ***bi***  чине параметре тих ***W.*** “



Илустрација 8: Архитектура повезане неуронске мреже и неурона

Израчунавања која мрежа извршава састоји се од линеарних матричних трансформација и примена активационе функције. У последњем слоју мреже, могуће је да функција 𝑔˜ може да представља другачију активациону функцију од фукције 𝑔. Последњи слој има само један неурон са вредношћу оцене циљне промњиве. Случај класификације је мало другачији, последњи слој има онолико неурона колико има класа и да се инстанца класификује у класу која припада неурону са највећом вредношћу. Последњи слој користи следећу векторску активациону функцију:

Код бинарне класификације уместо два излаза и функције довољно је користи један излаз и сигмоидну функцију. Архитектура таквог типа спада у логистичку регресију над претпоследњим слојем мреже. У зависности од саме примене, потребно је поставити одговарајућу функцију грешке. На пример, случају регресије то би била квадратна грешка:

Када нам је потребна метода класификације одређеног броја класа, прво треба одредити одговарајућу репрезентацију категоричке променљиве. Обично таква презентација подразумева да је класа дефинисана вектором ***y*** димензије ***m***, који има ***m-1*** нула и једне јединице, при чему јединица на позицији ***ј*** ако и вектор ***y*** представља класу ***ј.*** Свака инстанца има један такав вектор ***y*** у зависности којој инстанци припада. Функција грешке која се највише користи у овом случају за категорилке циљне промењиве која се ослања на ту репрезентацијз је **унакрсна ентропија**.

Уколико желимо да неуронска мрежа врати жељене резулате и буде успешна, потребно је дефинисати и функцију грешке., а потом ускладити одговарајући алгоритам учења. Алгоритам који се користи за израчунавање градијента у случају неуронских мрежа назива се алгоритмом **пропагације уназад**. Алгоритам за израчунавање градијента се просто састоји од множења ивода тих активационих функција и компатибилних матрица и таблицом извода сложене функције. Проблем дубоких мрежа је у томе што њихове велике дубине могу довести до великог броја процеса, односно множења. Главна предност потпуно повезаних мрежа је у томе што су „агностичке структуре“, тј. нема посебних претпоставки о улазу. Иако структуре без претпоставке чини потпуно повезане мреже веома широко применљивим, такве мреже имају тенденцију да имају слабије перформансе од мрежа са посебном применом. Иако потпуно повезане мреже не дају никакве претпоставке о улазу, оне обично имају мање перформансе и нису добре за екстракцију карактеристика. Осим тога, имају већи број тежина за тренирање што резултира великим временом тренинга.

Код потпуно повезаних мрежа број улаза је унапред одређен и такве мреже имају ограничену примену у пракси. Највише се користе над подацима у виду вектора фиксне дужине. Из таквих разлога, потпуно повезане неуронске мреже тешко је применити над подацима попут слика различитих размера и секвенцама које се користе у биоинформатичким истраживањима. Када говоримо о подацима у табеларним формама, потпуно повезане мреже је могу дати изузетне резултате.

# Конволутивне неуронске мреже

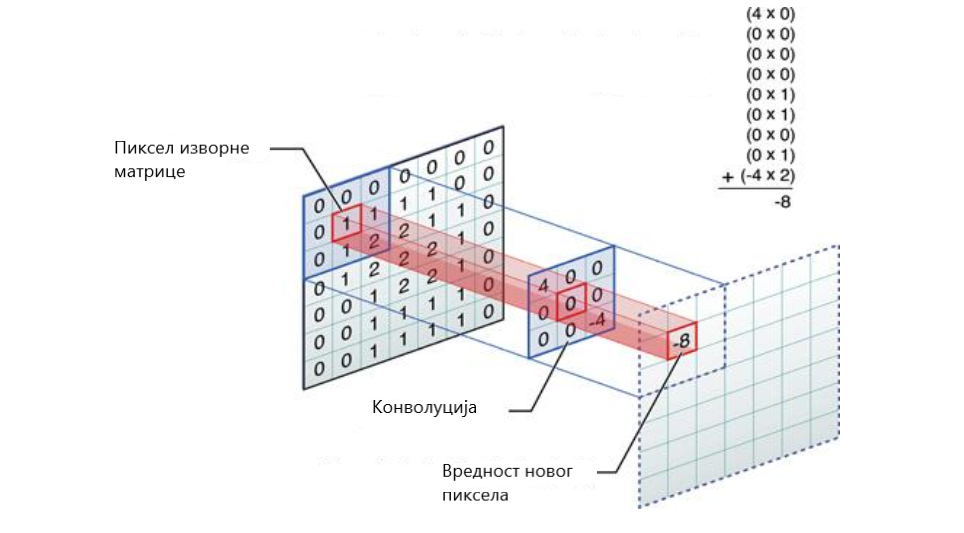
Конволутивне неуронске мреже су веома сличне порпуно повезаним неуронским мрежама: ​​састоје се од неурона који имају своје тежине и претпоставке које се могу научити. Сваки неурон прима своје улазе, изводи тачкасти производ и опционо га прати са нелинеарношћу. Цела мрежа и даље изражава једну диференцибилну функцију резултата: од необрађених пиксела слике на једном крају до резултата класе на другом. И још увек имају функцију губитка као код потпунo повезаних неуронских мрежа (нпр. Softmax) на последњем (потпуно повезаном) слоју и све што смо до сада рекли за учење потпуно повезаних мрежа и даље важи. Захваљујући оваквој архитектури, догодили су се велики помаци у сфери дубоког учења. Сама жеља за њиховим учењем дошла је из потребе за практичном применом над подацима у виду слика. На пример, за одређивање и препознавање објеката на сликама.

Потпуно повезане неуронске мреже не скалирају добро величину слике. У *CIFAR-10* скупу података, који је најчешће коришћен скуп у машинском учењу, слике су димензија 32х32х3. Што би значило да су слике слабије резолуције (32 широка, 32 висока и 3 канала у боји). Такав скуп нам дозвољава да ефикасно примењујемо различите алгоритме у зависности од проблема који треба решити. Узимајући овај случај, порпуно повезани неурон у свом скривеном слоју имао би тежину која би износила вредност од 3072 (32\*32\*3 = 3072). Јасно је да се овом тежином може управљати, али ова потпуно повезана структура се не скалира на веће слике. Уколико би слика имала димензије 200х200х3, то би значило да би неурон имао тежину од 120,000 (200\*200\*3). Са таквим начином, неурони би се преоптеретили и не би добили добре резулате. Закључак се намеће да је потпуно повезаност нескалабилна и велики број параметара би довео до преоптерећења (*overfitting*). Зато су нам неопходне методе трансформације.

Трансформације су методе које од слика производе нову слику истих или сличних димензија. То значи да можемо променити различите параметре слика као што су димензије, ивице, осветљење, контраст, изоштреност, итд. Производ таквих трансформација су стандардизоване слике, које се могу користити као својства на основу које би модел могао да учи. Различите трансформациј слика могу се вршити операцијама са две матрице. Једну матрицу би одредили као улазни сигнал, односно слику, а друга матрица би користили као филтер, који одређује процес који треба да се изврши.

Дефиниција конволуције матрица: „Конволуција матрица А ∈ R*m x n*  и B ∈ Rp x q је матрица димензија a (𝑚−𝑝+ 1)×(𝑛−𝑞 + 1) која означава производ А \* B и дефинише на следећи начин:

Конволуција је матричнa операцијa, која се састоји од језгра, матрице малих тежина, која прелази преко улазних података вршећи множење по елементима са делом улаза на коме се налази, а затим сабира резултате у излаз. У математичкој анализи, конволуција је математичка операција са две функције (f  и g) која производи трећу функцију (f \* g). Општи облик за матрицу конволуција је дат тако што, конволуција омогућава дељење тежине. То омогућава смањење броја ефективних параметара и дозвољава да се исти параметар слике открије у различитим деловима улазног простора. Оваква метода конволуције је кључна тачка конволутивних неуронских мрежа.



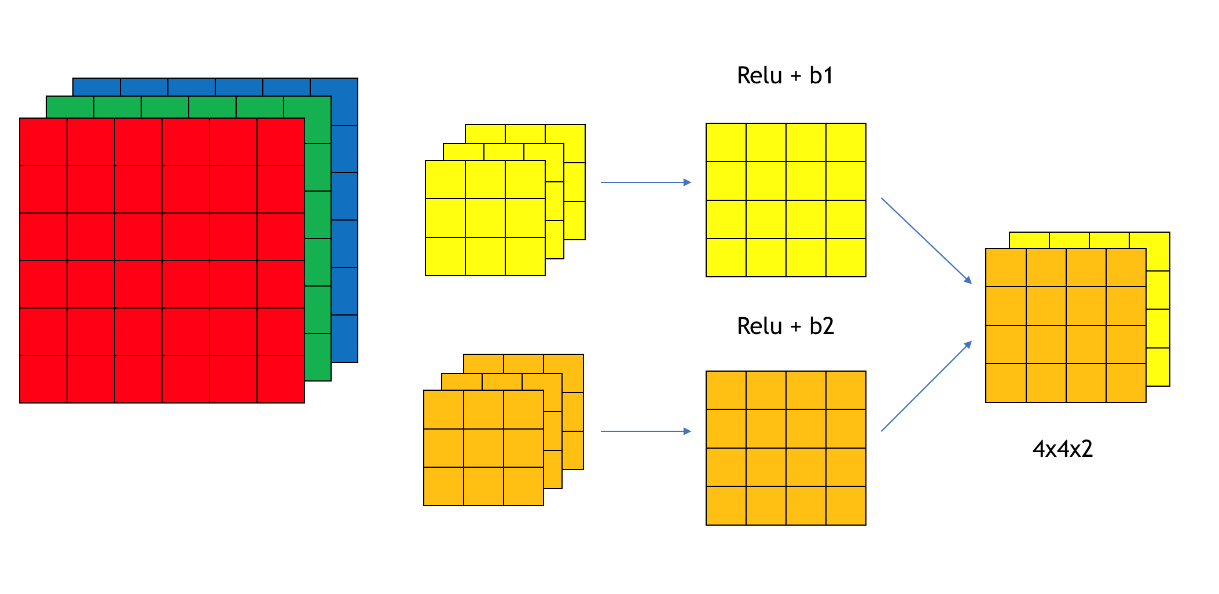
Илустрација 9: Конволутивне операције над матрицама

У општем случају матрице не морају бити квадратне. Конволуција за свако **𝑖** и **𝑗** израчунава скаларарни производ елемената матрице ***B*** са одговарајућим деловима матрице ***А*** по формули. Матрица ***В*** помера се дуж матрице  ***А*** и рачуна скаларне производе на различитим пољима и на тај начин добијамо резултат конволуције.

Често се може догодити да ни ексеперти који праве избор слика и дизајнирају релевантне трансформације, нису довољно компетентни да дефинишу довољно добре филтере за неке одређене задатке. Идеја која би могла да реши такву проблематику, имплементира се тако што би се модел могао дефинисати тако да свој саставни део научи и трансформише, а не само да донесе закључак на основу излаза. То би се могло дефинисати основна идеја конволутивних мрежа. Матрицу ***В*** посматрамо као филтер приликом израчунавања конволуције, која се креће дуж матрице ***А*** и на тај начин добијамо скаларни производ на различитим локацијама. Сваки скаларни производ могуће је узрачунати засебним неуроном. На такав начин могуће је применити процес конволуције на групи неурона. Оно што је битно, то је да сви неурони из те групе морају имати исте параметре филтера.

Конволутивна мрежа смањење грешака својих излаза на датим улазим постиже обучавањем. Филтери конволутивне мреже могу бити научени, што значи да би било потребно дефинисати архитектуру и структуру мреже која то може успешно да реши. Улази се трансформишу низом конволуција, активационим функцијама и агрегације које се међусобно преплићу, а да је производ тога порпуно повезана неуронска мрежа, која може да врши завршну класификацију улаза. Да би процес конволуције у неуронским мрежама био успешан, неопходно је имати више улазних канала. На тај начин, улаз не представља увек матрицу већ може представљати тродимензионални низ који представља неку врсту тензора. Једноставније је тродимензионални низ представити као колекцију дводимензионалних тензора тј. матрица, које називамо каналима. Сада, улазни тензор и филтер биће тродимензионални тензор, а излаз ће бити представљен као обична матрица.

Неуронска мрежа у сваком свом слоју учи одређени број матрица које представљају филтере. Слој који чини већи број филтера називамо конволутивни слој мреже. Конволутивни слој даје као излаз тензор који одговара излазу појединачног филтера. Сваки филтер има своју функцију и параметре по којима врши филтрацију, нпр. Један филтер служи да уочи вертикалне ивице, други хоризонталне и трећи косе. Нелинеарном активационом функцијом се трансформише сваки канал као и код порпуно повезаних неурноских мрежа. После једног конволутивног слоја могуће је надовезати наредни слој. Тако да виши конволутивни слојеви процесирају податке које су процесирали нижи слојеви.



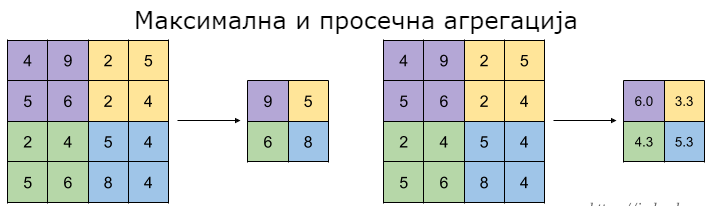
Илустрација 10: Конволутивни слој

Поред конволутивног слоја потребно је споменути и слој агрегације (*pooling layer*). Овај слој има главну улогу у удруживању параметара у циљу смањења количине израчунавања и учинили неке од детекција више робусним. Реализација решења је једноставна. Потребно је да сваки канал улазног тензора подели на делове одређених димензија који се замене једном вредношћу попут максимума вредности у том тензору. Производ високих вредности у тензору значи да је филтер који је произвео ту вредност када је нашао образац који тражи. Процесом агрегације губи се информација на ком месту се налазио максимум. Колики је то значај те информације, зависи искључиво од премене.

Сада ћемо процес агрегације објаснити преко примера. Процес је прилично једноставан. Улаз који је приказан има димензију 4х4 и на слици су приказана са процесом рачунања максимума и просека. У примеру, филтер има димензију 2х2 и помера се за два поља, због тога је потребно улазну матрицу раставити на 4 улаза са димензијом од 2х2. За добијање максималне агрегације, потребно је израчунати максималну вредност улаза из сваке под регије. Принцип за добијање просечне агрегације је исти само је потребно израчунати просечну вредност. Разлика између конволуције и агрегације је та да се максимум не рачуна на преклапајућим деловима, већ се дели на дисјунктивне подрегије који одговарају димензијама агрегације. Када улазни слој има више од једног канала, сваки слој се обрађује независно. У овом примеру је свака димензија слике дељива одговарајућом димензијом слика (*n x n*). У случају када не би била дељива димензија слике са димензијом филтера, колона или врста која представља остатак тог дељења не би била обрађена. На пример, када је димензија улазног канала 4х3, а филтер димензије 3х3, последња врста би била занемарена.

Дефиниција конволутивне мреже је функција 𝑓w која је одређена на следећи начин:

где ***х*** представља улазни тензор, ***𝐿*** је број конволутивних слојева, ***h𝑖***  за ***𝑖 = 1, . . . ,𝐿*** представља излазни тензор ***𝑖***'-тог слоја, ***W𝑖***  је тензор параметара ***𝑖***'-тог слоја***, B𝑖*** тензор слободних коефицијената који по димензијама одговара тензору ***W𝑖 \* h𝑖−1***, ***𝑎***  је функција агрегације, ***𝑔*** је активациона функција***, ˜𝑔˜w*** је потпуно повезана неуронска мрежа.



Илустрација 11: Агрегација максималне и просечне вредности

# Примењене методе у овом раду

Развоју апликације за класификацију рендгенских налаза плућа могли би смо поделити у четири фазе. Прва од четири фазе је сакупљање и обрада података, односно приказивање и анализа података кроз графике. Друга фаза се састоји од креирања и тренинга модела за класификацију слика. У трећој фази приступамо методи пренесеног учења, користећи унапред обучени модел са замрзнутим слојевима, што значи да ће тежине остати не промењене. Четврта фаза рада је прецизно подешавање. У овој фази последњи слојеви претходно обученог модела нису замрзнути. Програмски код је писан је у програмском језику Пајтон (Python) користећи Керас (Keras) и остале библиотеке.

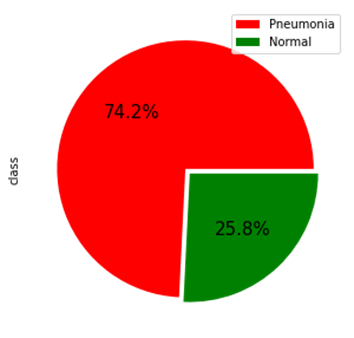
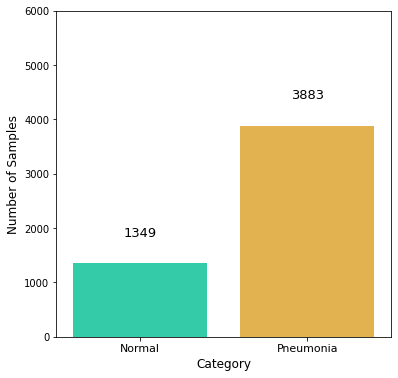
## Прикупљање података

Овај сет података садржи 5,856 рендгенских снимака у облику слика. Слике су подељене на тренинг и тест сет података од различитих пацијената. Слике у свом називу имају врсту упале као на пример: *NORMAL/BACTERIA/VIRUS,* затим насумично додељен идентификациони број пацијента и број слике пацијента.

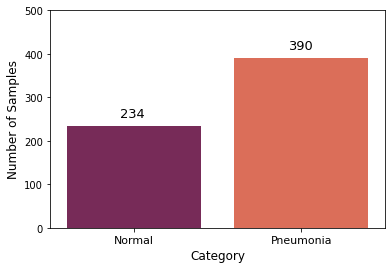
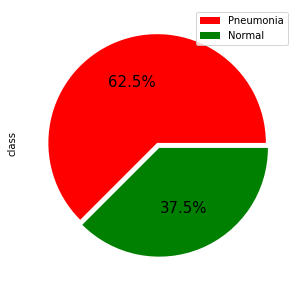
Рендгенске слике грудног коша одабране су из ретроспективних група педијатријских пацијената. Ови рендгенски снимци рађени су као део рутинске контроле и клиничке неге пацијента.

Сви рендгенски снимци су првобитно испитани и прегледани ради контроле квалитета и уклоњни су сви снимци ниског квалитета. Дијагнозе на основу слика су успоставили два стручна лекара пре него што је добијена дозвола за коришћење и обуку у интелигентним системима. Како би отклонили могућност грешке, било је потрено да провери сет података и трећи стручни лекар.

Ови подаци се користе како би се развио модел који омогућава бинарну класификацију предикцијом за свако патолошко стање. Овај сет података је распоређен у фолдере за тренирање модела који се састоји од 5,232 слике распоређене у два фолдера под називом *NORMAL* и *PNEUMONIA.* У фолдеру *NORMAL* налази се 1,349 слика које представљају плућа без упале, док у фолдеру *PNEUMONIA* се налази 3,883 слика које представљају плућа са видљивим симптомима упале плућа. Фолдер за тестирање модела садржи 624 слике које су распоређене такође у два фолдера под називом *NORMAL* и *PNEUMONIA* вођене истим принципом расподеле.

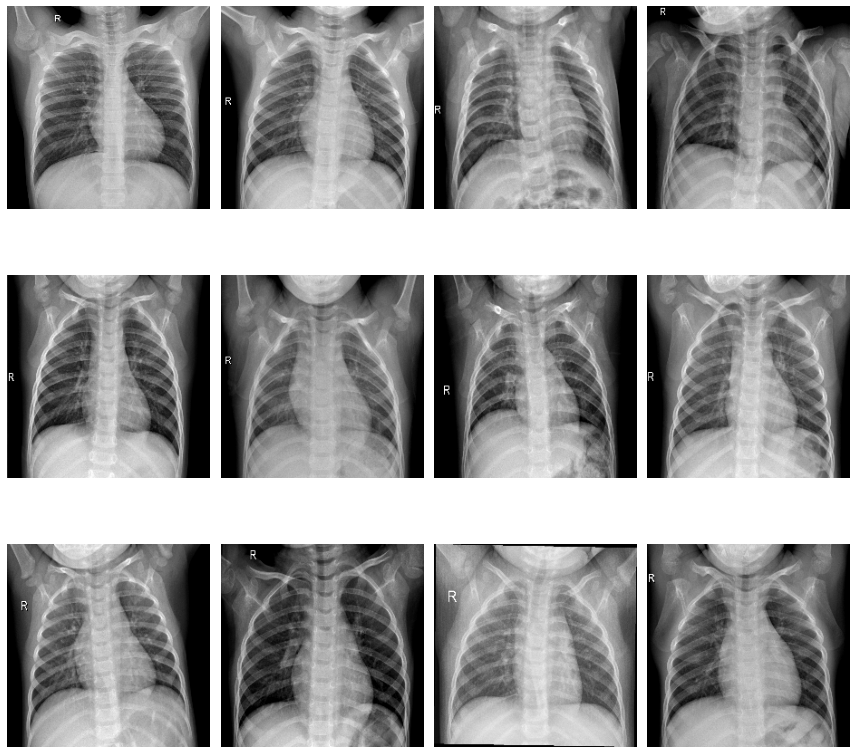


Илустрација 12 и 13: Приказ броја узорака и однос категорије NORMAL и PNEUMONIA у проценитима из фолдер за тренирање мoдела

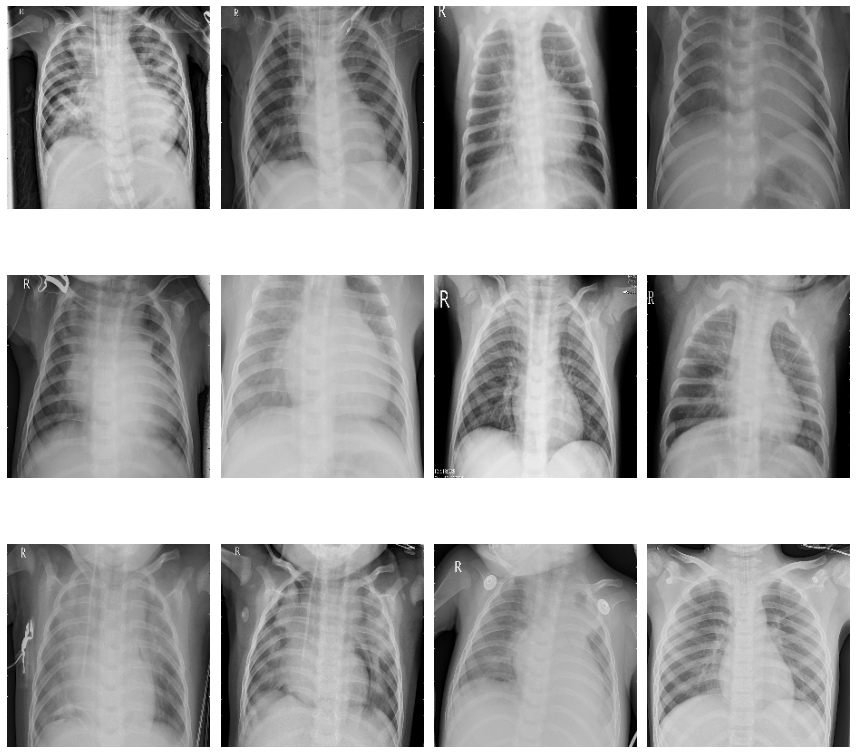
 

Илустрација 14 и 15:Приказ броја узорака и однос категорије NORMAL и PNEUMONIA у проценитима из фолдер за тестирање мoдела

На овим приказима можемо закључити да ови скупови података се мало разликују једно од других. Оба скупа су мало неуравнотежена, постоји више узорака из категорије са пнеумонијом, уз то да је скуп података за тренирање модела мало више неуравнотежен. На следећим сликама приказани су узорци из оба скупа података.



Илустрација 16: Узорци из скупа података за тренирање модела без упале



Илустрација 17: Узорци из скупа података за тренирање модела са упалом

# Припрема података

Пре креирања и тренирања модела за класификацију слика, потребно је прво формирати скуп за валидацију. Потребно је своје улазне податке расподелити у подскупове за обуку, валидацију и тестиранње како би се спречио *overfitting* у процесу учења и да би могли да ефикасно проценимо наш модел. *Оverfitting* се дешава када модел научи претерано добро до те мере да негативно утиче на перформансе модела на новим подацима. То значи да се лоши делови узорка науче као што су шум и остале нестабилности у подацима и користе као ваљани концепти у оквиру модела. Овакав начин процеса негативно утиче на способност модела да се генерализују. Вероватноћа за оваквом појавом је већа код непараметарских и нелинеарних модела који имају већу флексибилност приликом учења циљне функције. Како би смо боље схватили разлику између скупа за тренинг, валидацију и тестирање, треба обратити пажњу на одломак из текста из Риплијеве књиге „Препознавање узорака и неуронске мреже“ (Pattern Recognition and Neural Networks). Ови скупови податак су дефинисани на следећи начин:

„*– Сет за обуку: Скуп примера који се користе за учење, односно да одговарају параметрима класификатора.*

*– Скуп за валидацију: Скуп примера који се користе за подешавање параметара класификатора, на пример за избор броја скривених јединица у неуронској мрежи.*

*– Тестни скуп: Скуп примера који се користи само за процену перформанси потпуно одређеног класификатора.“ 1*

У овом случају користимо класичну поделу оригиналог скуп података за тренирање. Делимо скуп на 80% за стварну обуку и 20% за потребе валидације. Затим је потребно учитати слике из фолдера и припремити их за обраду.

Почињемо са одређивањем генератора података. Са *Keras Image Data Generator*, можемо променити и прилагодити величину пиксела и применити насумичну трансформацију за повећање података. Користимо два различита генератора, један за тренинг – *train\_imggen*, други за валидацију – *val\_imggen*.

Ове генераторе примењујемо на сваки скуп података како би све слике биле стандардизоване, помоћу функције *flow\_from\_dataframe*. Ова метода је склопу *Keras* библиотеке, али је уз то потребно и инсталирати *Pandas* библиотеку. Осим трансформација дефинисаних у сваком генератору, слике се такође мењају на основу параметара у *target\_size.* Величина слике постављена је на 224 (*image\_size* = 224), *batch* износи 32, а *seed* 42.

# Дефинисање модела

После припреме података спремни смо на следећу фазу пројекта, а то је креирање и обучавање модела за класификацију слика.

Пре креирања модела, потребно је поставити повратне позиве односно *callbacks.* Креирање модела без њих је као вожња возила без функционалних кочница. Имамо малу или никакву контролу над целим процесом који ће вероватно довести до нежељењног исхода.

Из *Keras* документације можемо дефинисати: „*Повратни позив (callback) је скуп функција које се примењују у датим фазама процеса обуке. Можете користити повратне позиве да бисте добили преглед интерних стања и статистике модела током обуке.“ 2*

Повратни поозиви (*callbacks*) највише користимо када желимо да аутоматизујемо неке процедуре након сваке обуке или епохе и помажу вам да имате контролу над целокупним процесом обуке. То значи да имате бољи увид када је неопходно зауставити тренинг када се достигне конкретна вредност за тачност или за губитак функције, затим након сваке успешне епохе чува се одређена вредност контролне тачке, прилагођава се стопа учења током процеса и још много тога. У овом пројекту користио сам *EarlyStopping* функцију. Овој функцији можете доделити различите параметре и аргументе које се могу мењати у зависности када је погодно да се процес обуке заустави. Неке од метрика које се користе су:

*monitor*: вредност коју је потребно пратити - val\_loss

*patience*: број епоха без побољшања након којих ће обука бити прекинута, у нашем случају је то 5 епоха

*min\_delta:* минимална промена у надгледаној вредности. На пример, *min\_delta* =1 значи да ће процес обуке бити заустављен ако је апсолутна промена надгледане вредности мања од 1, код нас је доста мања вредност 0.0000001

*restore\_best\_weights:* подешавамо ову метрику на Тачно (*Тrue*) ако желиmo да задржиmo најбоље тежине када се заустави процес

Други повратни позив је *ReduceLROnPlateau*. Овај повратни позив смањује стопу учења када метрика престане да се побољшава. Такође се прати количина и ако се не види напредак за одређен број епоха, стопа учења се смањује.

# Креирање модела

После успешног дефинисања модела, потребно је креирати модел за класификацију слика. У овом делу пројекта модел је пројектован од нуле. Први слој модела је улазни слој који је дефинисан и има за аргументе висину и ширину од 224 пиксела и 3 канала тј. три боје (црвена, зелена плава), односно формат слике је 224 х 224 *RGB (red, green, blue).* У овом случају је укупан број улаза 150 528 (224 х 224 х 3) за улазни слој. Може се користити и аргумент *None,* у колико димензија има променљиву величину. Вишеслојне неуронске мреже користе улазни вектор слике и трансформишу га кроз низ скривених слојева, користећи нелинеарне активационе функције. Сваки скривени слој је такође у потпуности повезан са свим неуронима у претходном слоју. Последњи слој је такође потпуно повезан и представља коначну излазну класификацију. Потребно је додати сакривене слојеве у мрежу. Постоји много типова слојева који се користе за изградњу конволутивне мреже, али они који су се користили у овом моделу подељени су у три блока и то су следећи слојеви:

* Конволутивни слој – користили смо *Keras* методу *Conv2D*(). Конволутивни слој је основни део блока конволутивне неуронске мреже. Аргументи ове методе су *filters, kernel\_size* и *padding. Filters* представља број излазних филтера у конволуцији, овом примеру је вредност 16 у првом блоку, у другом је 32 и у трећем 64. *Kernel\_size* је цео број или листа од два цела броја, који одређују висину и ширину прозора 2Д конволуције. Може бити један број за исте вредности просторне димензије, код нас је вредност 3. *Padding* може имати вредности *“valid”* и *“same”.* Вредност у овом моделу је *“valid”* што значи да нема допуњавања.
* Активациони слој – користили смо *Keras* методу *Activation()*. Као активациону функцију користили смо ‘*relu*’.
* Aгрегациони слој – дефинисали смо овај слој методом *MaxPool2D*(). Узима максималну вредност преко улазног прозора за сваки канал улаза. Прозор се помера корацима дуж сваке димензије. Пошто смо користили у конволутивном слоју *“valid”* за вредност *padding* онда није било потребе навести димензије овог прозора.
* *Batch-normalization* слој – у овом слоју се нормализују улази. Примењује трансформацију која одржава средњи излаз близу нуле и стандардну девијацију излаза близу 1.
* *Dropout* слој – насумично поставља улазне јединице на 0 са фреквенцијом стопе (*rate*) на сваком кораку током тренинга, што помаже у спречавању прекомерног прилагођавања (*overfitting).* Улази који нису подешени на нулу се повећавају за 1/(1 – *rate)* тако да збир свих улаза остане непромењен.
* *Danse* слој – густо повезани слој неуронске мреже, који имплементира функцију: ***output = activation(dot(input, kernel) + bias)***, при чему је *activation* активациона функција, *kernel* је матрица тежина коју креира слој, а *bias* је креиран вектор пристрасности према слоју (примењиво само ако је *use\_bias* постављен на тачно).
* *Flatten* слој – пре него што подаци пређу из предзадњег слоја у задњи слој, морају се довести у једнодимензионални слој.

Модел можемо дефинисати као функцију *get\_model()* у коду:

def get\_model():

    #Input shape = [width, height, color channels]

    inputs = layers.Input(shape=(IMG\_SIZE, IMG\_SIZE, 3))

    # 1st Block

    x = layers.Conv2D(filters=16, kernel\_size=3, padding='valid')(inputs)

    x = layers.BatchNormalization()(x)

    x = layers.Activation('relu')(x)

    x = layers.MaxPool2D()(x)

    x = layers.Dropout(0.2)(x)

    # 2nd Two

    x = layers.Conv2D(filters=32, kernel\_size=3, padding='valid')(x)

    x = layers.BatchNormalization()(x)

    x = layers.Activation('relu')(x)

    x = layers.MaxPool2D()(x)

    x = layers.Dropout(0.2)(x)

    # 3rd Block

    x = layers.Conv2D(filters=64, kernel\_size=3, padding='valid')(x)

    x = layers.Conv2D(filters=64, kernel\_size=3, padding='valid')(x)

    x = layers.BatchNormalization()(x)

    x = layers.Activation('relu')(x)

    x = layers.MaxPool2D()(x)

    x = layers.Dropout(0.4)(x)

    # Head

    x = layers.Flatten()(x)

    x = layers.Dense(64, activation='relu')(x)

    x = layers.Dropout(0.5)(x)

    #Output Layer

    output = layers.Dense(1, activation='sigmoid')(x)

    model = keras.Model(inputs=[inputs], outputs=output)

    return model

Сажетак овог модела можемо видети на следећој слици. Овај део је користан да би имали комплетан увид у садржај и изглед овог модела са тачним бројем параметара који треба да се израчунају пре почетка тренинга. Позивањем функције *model.summary()* добили смо следеће резултате.

**Model: "model"**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Layer (type) Output Shape Param #**

**=================================================================**

**input\_1 (InputLayer) [(None, 224, 224, 3)] 0**

**conv2d (Conv2D) (None, 222, 222, 16) 448**

**batch\_normalization (BatchN (None, 222, 222, 16) 64**

**ormalization)**

**activation (Activation) (None, 222, 222, 16) 0**

**max\_pooling2d (MaxPooling2D (None, 111, 111, 16) 0**

**)**

**dropout (Dropout) (None, 111, 111, 16) 0**

**conv2d\_1 (Conv2D) (None, 109, 109, 32) 4640**

**batch\_normalization\_1 (Batc (None, 109, 109, 32) 128**

**hNormalization)**

**activation\_1 (Activation) (None, 109, 109, 32) 0**

**max\_pooling2d\_1 (MaxPooling (None, 54, 54, 32) 0**

**2D)**

**dropout\_1 (Dropout) (None, 54, 54, 32) 0**

**conv2d\_2 (Conv2D) (None, 52, 52, 64) 18496**

**conv2d\_3 (Conv2D) (None, 50, 50, 64) 36928**

**batch\_normalization\_2 (Batc (None, 50, 50, 64) 256**

**hNormalization)**

**activation\_2 (Activation) (None, 50, 50, 64) 0**

**max\_pooling2d\_2 (MaxPooling (None, 25, 25, 64) 0**

**2D)**

**dropout\_2 (Dropout) (None, 25, 25, 64) 0**

**flatten (Flatten) (None, 40000) 0**

**dense (Dense) (None, 64) 2560064**

**dropout\_3 (Dropout) (None, 64) 0**

**dense\_1 (Dense) (None, 1) 65**

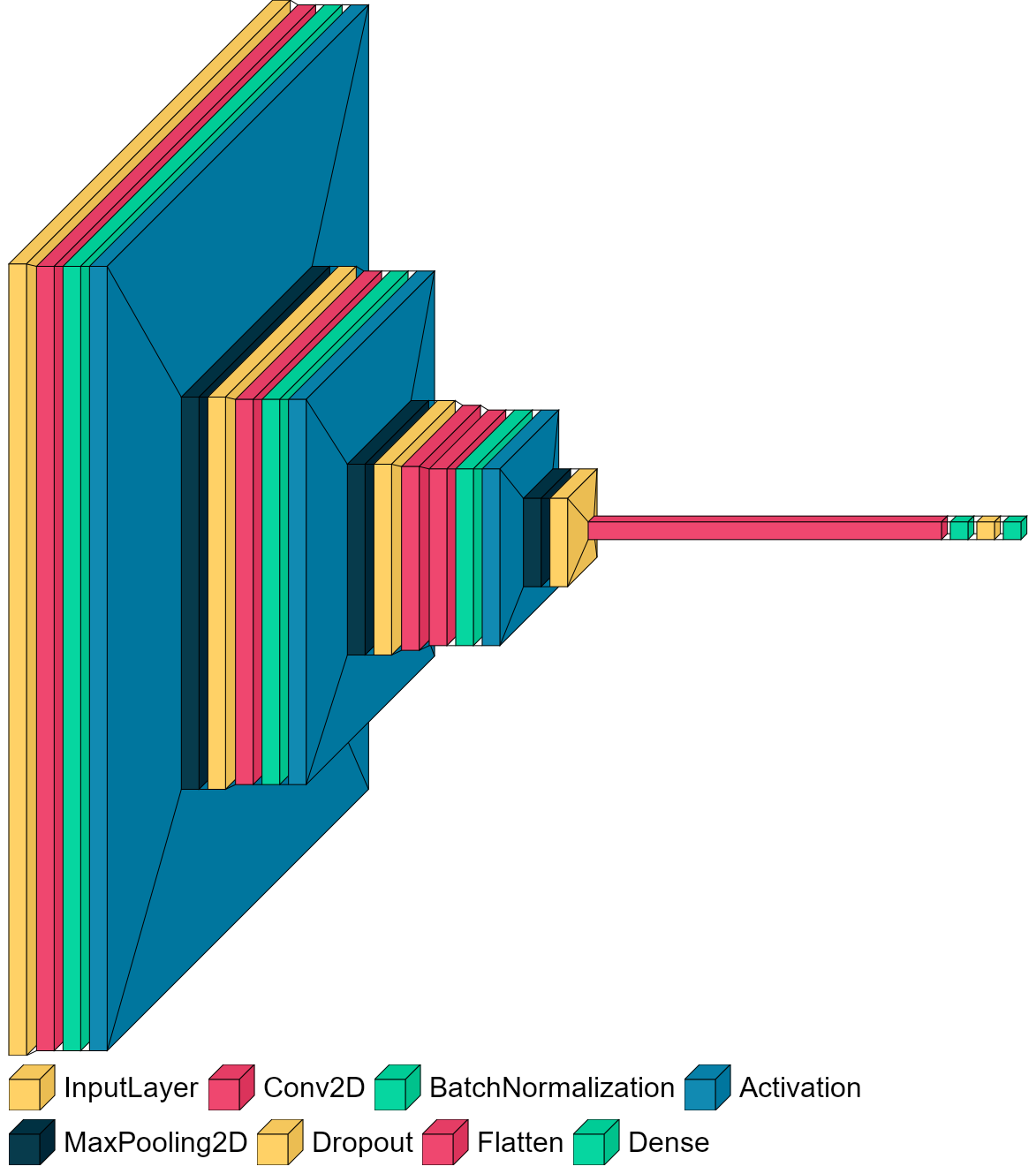
**=================================================================**

**Total params: 2,621,089**

**Trainable params: 2,620,865**

**Non-trainable params: 224**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**



Илустрација 18: Графички приказ модела

Након креираног модела и припреме потребних податак потребно је дефинисати губитке у овом процесу и изабрати адекватну „*Loss“*  функцију. Губитак се израчунава да би се добио градијент у односу на тежине које има модел и како би се те тежине ажурирале у складу са процесом. Губитак треба да се израчуна и ажурира након сваке итерације све док та ажурирања не донесу боље резултате. Одабир „*Loss“*  функције потребно је добро размотрити као при архитектури модела. У *Keras* библиотеци, *Loss“*  функција се прослеђује током фазе компајлирања као што је приказано испод:

model.compile(loss='binary\_crossentropy'

              , optimizer = keras.optimizers.Adam(learning\_rate=0.00003), metrics='binary\_accuracy')

Ми смо се определили за „*binary\_crossentropy“* . Бинарна унакрсна ентропија ће израчунати унакрсне ентропије између предвиђених класа и правих класа. Сврха функције губитка је израчунавање количине коју модел треба да покуша да минимизира током тренинга.

# Тренирање модела

Сада, када смо креирали модел и доделили функцију губитка када смо компајлирали наш модел, спремни смо за тренинг мреже. Позивањем методе *fit*() почињемо процес обучавања модела тако што ће податке за тренинг (*train\_dataset*) сећи у серије (*„batches")* које су одређене величином те серије „*batch\_size“* и понављати процес за дати број епоха („*epochs“).* Поставили смо 50 епоха за ово обучавање, доделили смо сет података за валидацију, повратне позиве који смо у претходних корацима дефинисали, а број корака по епохи и кораке за валидацију смо израчунали дељењем величине скупа података са бројем серија (*BATCH*).

У коду можемо видети како је имплементиран овај процес:

history = model.fit(train\_dataset,

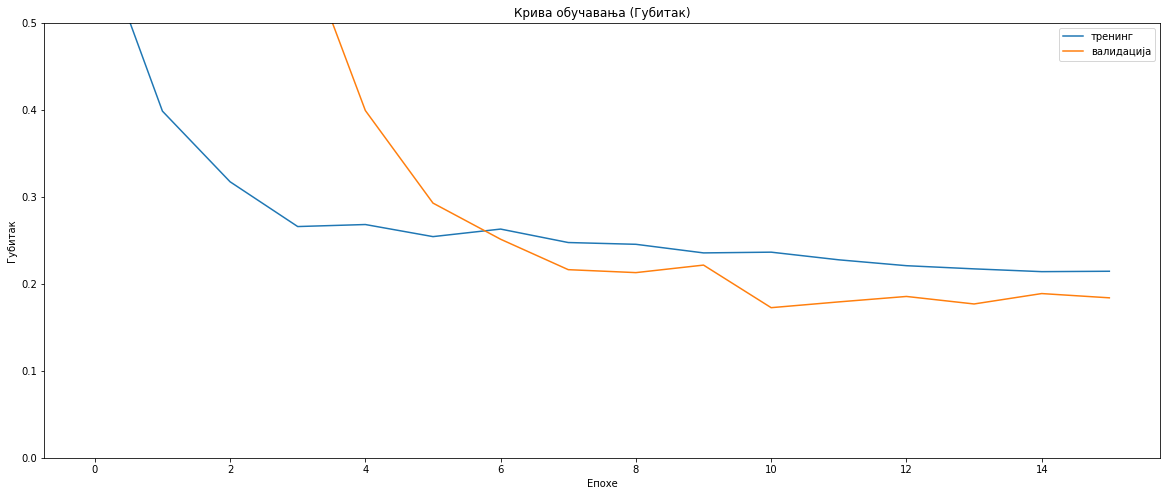
          batch\_size = BATCH, epochs = 50,

          validation\_data=val\_dataset,

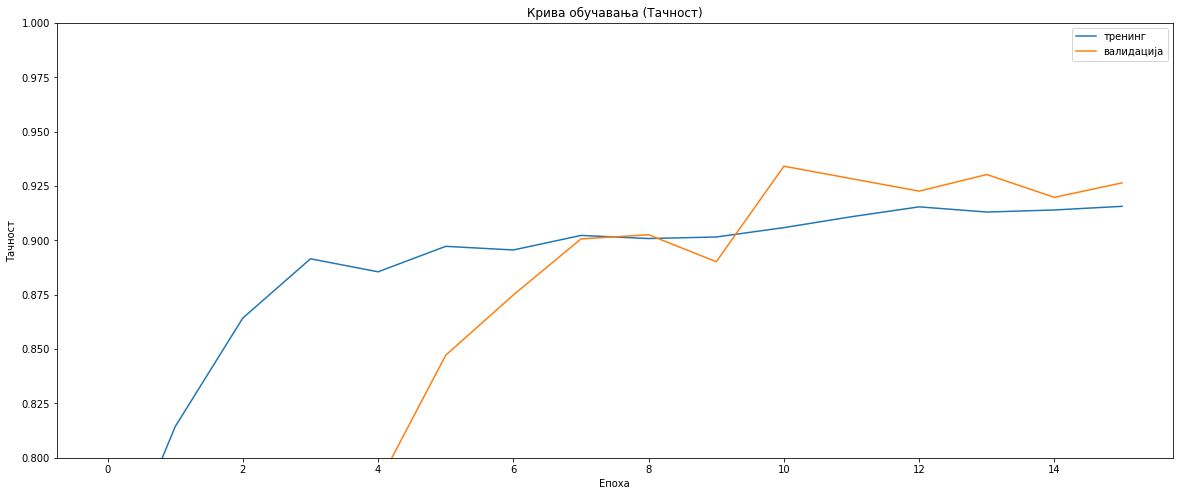
          callbacks=[early\_stopping, plateau],

          steps\_per\_epoch=(len(train\_df)/BATCH),

          validation\_steps=(len(val\_df)/BATCH));



Илустрација 19: Крива тренинга за губитак - „Loss”



Илустрација 20: Крива тренинга за тачност - „Accuracy”

Резултати обуке се могу видети на сликама 19 и 20 описани графиконима који представљају вредности за губитак и тачност у процесу тренинга и валидације. Разлика између вредности за тачност тренинга и валидације није велика, с тим да вредности за валидацију су веће него за тренинг (92,65 према 91,57%). Вредности за губитак су ниже за валидацију него за тренинг (0.1839 према 0.2145). Време извршења обучавања у просеку износи 60 минута.

После завршене обуке, потребно је тестирати модел и испитати његову тачност и губитке над скупу податак за валидацију и тестирање на следећи начин:

score = model.evaluate(val\_dataset, steps = len(val\_df)/BATCH, verbose = 0)

print('Val loss:', score[0])

print('Val accuracy:', score[1])

Добијени резултати за валидацију:

Val loss: 0.17259933054447174 Val accuracy: 0.934097409248352

score = model.evaluate(test\_dataset, steps = len(df\_test), verbose = 0)

print('Test loss:', score[0])

print('Test accuracy:', score[1])

Добијени резултати за тест скуп:

Test loss: 0.4277172088623047 Test accuracy: 0.8509615659713745

Из овога можемо видети да су резултати код скупа за валидацију бољи, они износе 0.1725 за губитак и 93.4% за тачност, док код скупа за тестирање 0.4277 за губитак и 85.09% за тачност.

# Пренесено учење

Други приступ који користимо назива се пренесено учење (*Transfer Learning*). Састоји се од унапред истренираног модела. Овакво учење модела се састоји од узимања функција научених на једном проблему и њихово коршћење на новом. Најчешћи кораци у имплементирању пренесеног учења су следећи:

* Коришћење слојева са претходно обученог модела
* Замрзавање истих ради избегавања уништења информација
* Додавање нових слојева који се могу обучити, научиће да претвори старе параметре у предности на новом скупу података
* Обучите нове слојеве на нашем скупу података

Модел који користимо је ResNet152V23 који је досупан у „*Keras“* библиотеци. Ово је модел који је већ обучен на другом скупу података (*ImageNet*). За параметар „***weights“* постављамо вредност „***ImageNet“ 4.* Покушали смо да изменимо крајњи слој мреже (*Head part*), који је имао функцију додељивања класа у другом скупу података, с тим да задржавамо остале слојеве и да поставимо негативну вредност за параметар *include\_top* што онемогућава потпуно повезан слој на врху мреже. Користимо наших последњих неколико слојева, укључујући онај који је коришћен за израду добијених израза. У коду је то имплементирано на следећи начин:

base\_model = tf.keras.applications.ResNet152V2(

    weights='imagenet',

    input\_shape=(IMG\_SIZE, IMG\_SIZE, 3),

    include\_top=False)

base\_model.trainable = False

def get\_pretrained():

    #Input shape = [width, height, color channels]

    inputs = layers.Input(shape=(IMG\_SIZE, IMG\_SIZE, 3))

    x = base\_model(inputs)

    # Head

    x = layers.GlobalAveragePooling2D()(x)

    x = layers.Dense(128, activation='relu')(x)

    x = layers.Dropout(0.1)(x)

    #Final Layer (Output)

    output = layers.Dense(1, activation='sigmoid')(x)

    model = keras.Model(inputs=[inputs], outputs=output)

    return model

При компајлирању модела изабрали смо за *„Loss”* функцију „*binary\_crossentropy“,* као и у претходном моделу. Представили смо сажет модел преко функције *model\_pretrained.summary().*

keras.backend.clear\_session()

model\_pretrained = get\_pretrained()

model\_pretrained.compile(loss='binary\_crossentropy'

              , optimizer = keras.optimizers.Adam(learning\_rate=0.00005), metrics='binary\_accuracy')

model\_pretrained.summary()

**Model: "model"**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Layer (type) Output Shape Param #**

**=================================================================**

**input\_1 (InputLayer) [(None, 224, 224, 3)] 0**

**resnet152v2 (Functional) (None, 7, 7, 2048) 58331648**

**global\_average\_pooling2d (G (None, 2048) 0**

**lobalAveragePooling2D)**

**dense (Dense) (None, 128) 262272**

**dropout (Dropout) (None, 128) 0**

**dense\_1 (Dense) (None, 1) 129**

**=================================================================**

**Total params: 58,594,049**

**Trainable params: 4,731,137**

**Non-trainable params: 53,862,912**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Користили смо исти број епоха и серија као и у претходном моделу, са истим повратим позивима и начин на који смо израчунали кораке по епохи и валидацији је идентични. Након додавања слојева на већ истренирани модел, потребно је истренирати модификовани модел. Тачност коју смо добили над сетом за тренинг износи 96.82%, док за тачност за валидацију износи 96.56%. Вредности губитака су 0.0974 за тренинг сет, а за валидацију износе 0.0988. Овог пута вредности на тренинг сету су боље него над подацима које никад није видела мрежа.

history = model\_pretrained.fit(train\_dataset,

          batch\_size = BATCH, epochs = 50,

          validation\_data=val\_dataset,

          callbacks=[early\_stopping, plateau],

          steps\_per\_epoch=(len(train\_df)/BATCH),

          validation\_steps=(len(val\_df)/BATCH));

Након завршене обуке потребно је тестирати модел на скупу података за валидацију и тестирање на следећи начин:

score = model\_pretrained.evaluate(val\_dataset, steps = len(val\_df)/BATCH, verbose = 0)

print('Val loss:', score[0])

print('Val accuracy:', score[1])

Добијени резултати за валидацију:

Val loss: 0.09814384579658508 Val accuracy: 0.9675262570381165

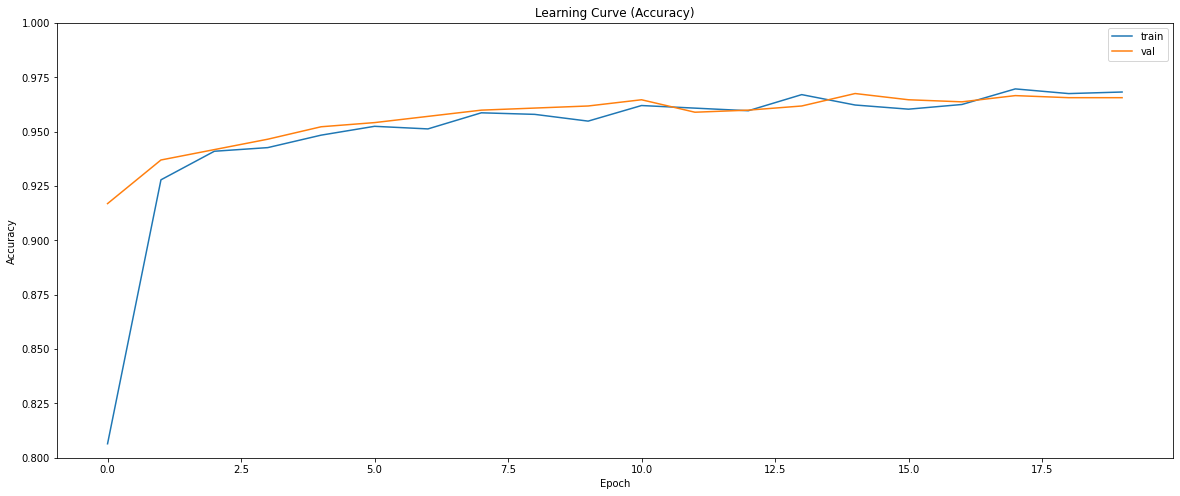
score = model\_pretrained.evaluate(test\_dataset, steps = len(df\_test), verbose = 0)

print('Test loss:', score[0])

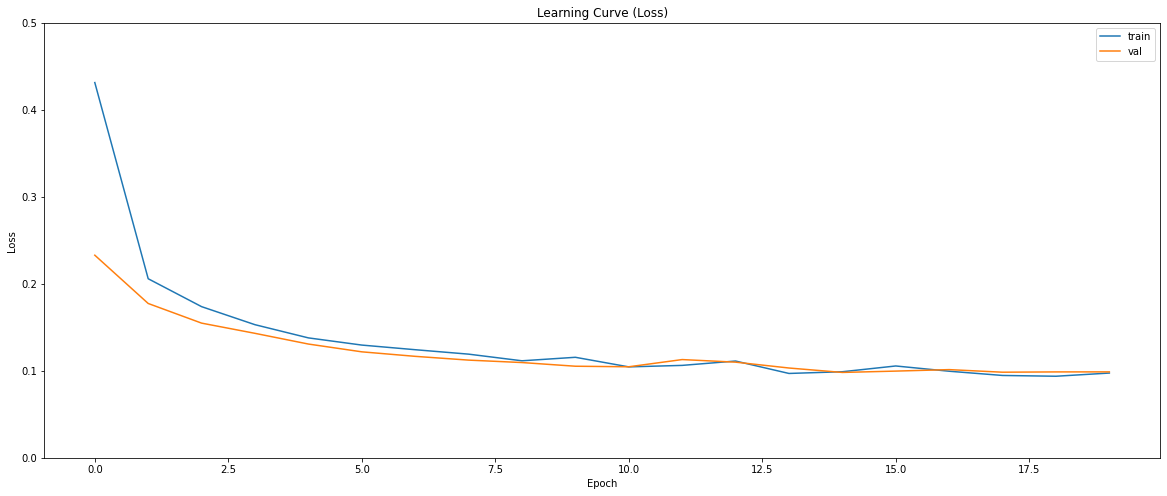
print('Test accuracy:', score[1])

Добијени резултати за тест скуп:

Test loss: 0.2969036400318146 Test accuracy: 0.870192289352417



Илустрација 21: Крива тренинга (Transfer learning) за тачност - „Accuracy”



Илустрација 22: Крива тренинга (Transfer learning) за губитак - „Loss”

Добијени резултати за овај модел бољи су него резултати претходног модела. Сада желимо да унапредимо мрежу трећим приступом. Тај приступ називамо „*Fine Tuning“*, односно прецизно подешавање.

# Прецизно подешавање

Наш последњи приступ се зове прецизно подешавање („*Fine Tuning*“). Код претходног приступа, пренесеног учења, било је потребно да се сви слојеви претходног модела „замрзну“ и на тај начин се сачувају тежине израчунате током обуке на другом скупу података (ImageNet). Сада желимо да „одмрзнемо“ неколико последњих слојева и да наставимо обучавање, прецизно подешавајући тежине тих слојева према нашем скупу података са ниском стопом учења. Овакав приступ је опцион и може потенцијално донети боље резултате и побољшање модела, али може довести до брзог преоптерећења мреже, то треба имати на уму. У овој фази, потребно је користити веома ниску стопу учења, јер се обучава много већи модел него у првом кругу обуке. Као резултат тога, ризикује се да се догоди „*overfitting“* , ако се примене велике тежине при ажурирању. У овој фази само желимо да унапред припремљене тежине.

Потребно је прво „*base\_model“* поставити да је могуће тренирати последњих 13 слојева.

base\_model.trainable = True

for layer in base\_model.layers[:-13]:

    layer.trainable = False

Позивањем методе „*compile()“* на моделу омогућава да „замрзне“ понашање тог модела. То значи да вредност атрибута који се могу обучити у време компајлирања да треба да буду сачувани, све док се функција компајлирања не позове поново. Ако се промени било која вредност која се може обучити, потребно је позвати поново методу „*compile()“* на моделу да би промене биле ажуриране.

model\_pretrained.compile(loss='binary\_crossentropy'

              , optimizer = keras.optimizers.Adam(learning\_rate=0.000002), metrics='binary\_accuracy')

model\_pretrained.summary()

Model: "model"

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Layer (type) Output Shape Param #**

**=================================================================**

**input\_1 (InputLayer) [(None, 224, 224, 3)] 0**

**resnet152v2 (Functional) (None, 7, 7, 2048) 58331648**

**global\_average\_pooling2d (G (None, 2048) 0**

**lobalAveragePooling2D)**

**dense (Dense) (None, 128) 262272**

**dropout (Dropout) (None, 128) 0**

**dense\_1 (Dense) (None, 1) 129**

**=================================================================**

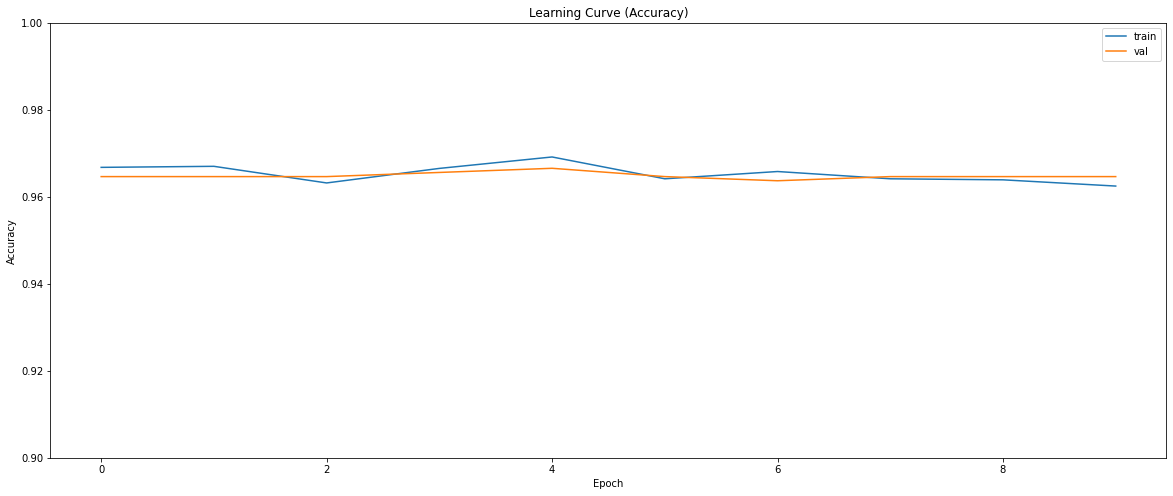
**Total params: 58,594,049**

**Trainable params: 262,401**

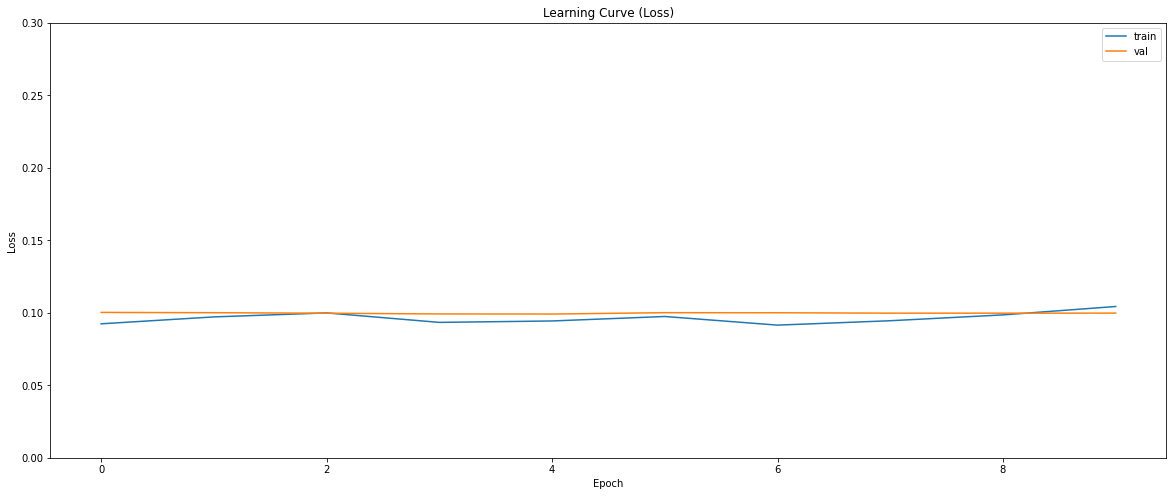
**Non-trainable params: 58,331,648**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Резултати обучавања су приказани на следећим графиконима:



Илустрација 22: Крива тренинга (Fine learning) за тачност - „Accuracy”

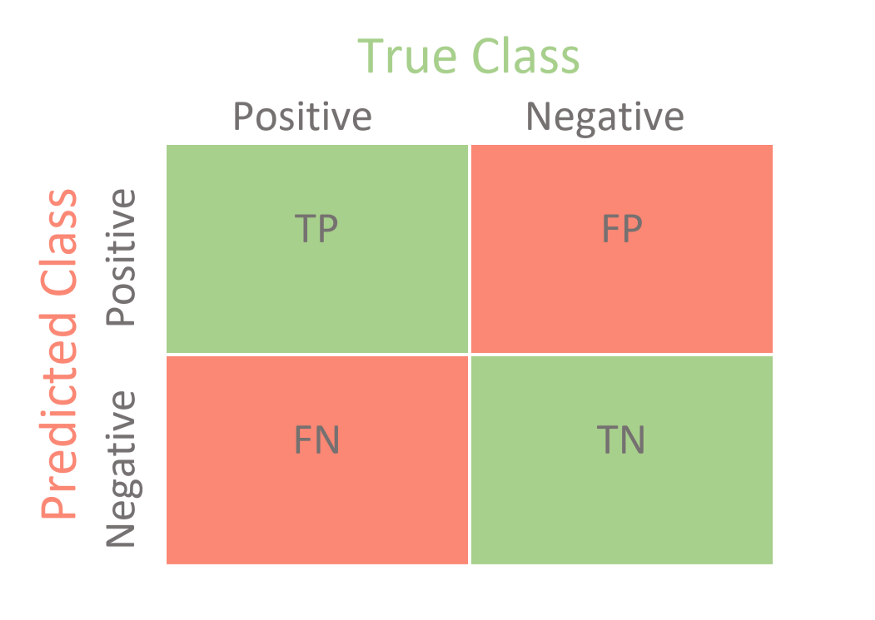


Илустрација 22: Крива тренинга (Fine learning) за губитак - „Loss”

Тачност модела је 96.78% на тренинг сету, док је за сет који је намењен за валидацију вредност износи 96.88%. Губитак који смо добили над сетом за валидацију износи 0.0998, а над тренинг сетом је губитак 1.044. Резултати тестирања за тачност су 87.98% и 96.78%, док су за губитак добијена вредност износи 0.099 за скуп података валидације и 0.280 за скуп намењен за тест.

# Поређење перформанси

Као што смо могли да претпоставимо, најбоље резултате донео је последњи приступ, Прецизно подешавање („*Fine Tuning“*). Вредност тачности при тестирању овог модела износи скоро 88%. Сада желимо да представимо перформансе модела „*Confusion“* матрицом. Резултат ове матрице даје поређење између стварних и предвиђених вредности у матричном облику. Матрица је *N x N,* што значи да је то квадратна матрица, где је *N* број класа или излаза. Ми смо имали пример са две класе, што значи да ће матрица имати димензију 2 х 2. „*Confusion“* матрица има четири појма који је чине, а то су: тачно позитиван (*True Positive-ТР*), лажно позитиван (*False Positive-FP*), тачно негативан (*True Negative-TN*), лажно негативан (*False Negative*-*FN*). Да би лакше разумели модел матрице потребно је погледати слику 23.



Илустрација 23: „Confusion“ матрица

Тачно позитивно (*FN*): Односи се на број предвиђања где класификатор тачно предвиђа позитивну класу као позитивну.

Тачно негативно (*ТN*): Односи се на број предвиђања где класификатор тачно предвиђа негативну класу као негативну.

Лажно позитивна (*FP*): Односи се на број предвиђања где класификатор погрешно предвиђа негативну класу као позитивну.

Лажно негативан (*FN*): Односи се на број предвиђања где класификатор погрешно предвиђа позитивну класу као негативну.



Илустрација 24: „Confusion“ матрица нашег модела

У овом случају матрица има вредности: тачно позитивни 166, тачно негативни 68, лажно позитивни 7, лажно негативни 383.

Након ових параметара, можемо добити извештај наше класификације помоћу методе „*metrics.classification\_report()“.*

print(metrics.classification\_report(Y\_test, pred\_labels, labels = [0, 1]))

**precision recall f1-score support**

**0 0.96 0.71 0.82 234**

**1 0.85 0.98 0.91 390**

**accuracy 0.88 624**

**macro avg 0.90 0.85 0.86 624**

**weighted avg 0.89 0.88 0.88 624**

Тачност („*Accuracy*“) представља број исправно класификованих инстанци података у односу на укупан број инстанци података. Дефинише се формулом:

Пошто нам тачност не представља једини валидан параметар потребно је пронаћи прецизност модела („*Precision*“). Прецизност би идеално требало да буде 1 (висока) за добар класификатор. Прецизност се дефинише на следећи начин:

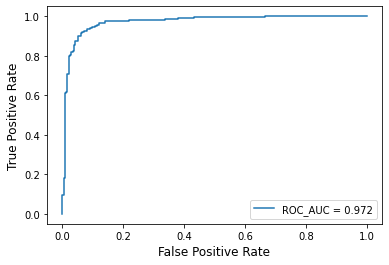
Прецизност постаје 1 само када су бројилац и именилац једнаки, тј. = +, то такође значи да је нула. Како се повећава, вредност имениоца постаје већа од бројилаца и вредност прецизности се смањује (што не желимо).

Сада уводимо још један важан параметар који се зове опозив („*Recall*“). „*Recall*“ је такође позната као осетљивост или тачно позитивна стопа („*true positive rate*“). Дефинише се формулом:

Дакле, идеално у добром класификатору, желимо да и прецизност („*Precision*“) и опозив („*Recall*“) буду један, што такође значи да су FP и FN нула. Због тога нам је потребна метрика која узима у обзир и прецизност и опозив. „*F1*“ резултат је метрика која узима у обзир и прецизност и опозив и дефинисана је на следећи начин:

„*F1*“ резултат постаје висок само када су и прецизност и опозив високи. „*F1*“ резултат је хармонична средина прецизности и опозива и боља је мера од тачности.

У машинском учењу мерење перформанси је суштински задатак. Дакле, када је у питању проблем класификације, можемо израчунати на „*AUC* – *ROC“* криву. Када треба да проверимо или визуелизујемо перформансе проблема класификације више класа, користимо „*AUC“* (Област испод криве) „*ROC“* (оперативне карактеристике пријемника) криву. То је једна од најважнијих метрика евалуације за проверу перформанси било ког модела класификације. Вредност коју смо ми добили помоћу методе „*metrics.roc\_auc\_score()“* износи 0.9719154065307911. Графички приказ криве предстаљен је на слици 25.



Илустрација 26: „AUC-ROC” крива

# Закључак

Због важности класификације медицинских слика и малог скупа података, у овом раду сам изабрао класификацију засновану на конволутивним неуронским мрежама. Користили смо скуп података са рендгенским сликама грудног коша. Коришћењем различитих приступа добија се најбољи резултат прецизним подешавањем („*Fine Tuning“).* Уопштено говорећи, методе конволутивних неуронских мрежа боље су од традиционалних метода јер могу да уче и бирају карактеристике ефикасно и аутоматски. Мрежа која је једноставна не може обично да научи довољно само из података, па због тога не постиже високу тачност. С друге стране, превише компликоване мреже теже се обучавају и могу да се брзо препуне, па прецизност остаје ниска. Модел који има одговарајућу димензију и ефикасне методе за спречавања оптрећивања мреже може да добије најбоље резултате. У нашој мрежи, „*recall*“ је био веома висок, скоро 100%. Постизање таквог резултата може се назвати веома добрим и показатељ је да је модел успешан, с обзиром да је скуп података за обуку био релативно мали.

Конволутивне неуронске мреже постигле су задивљујуће резултате различитим областима, укључујући медицинска достигнућа, а све је већи интерес примене у радиологији и патологији. Познавања кључних концепата и предности конволутивних неуронских мрежа као и ограничења дубоког учења је од суштинског значаја за коришћење ових технологија у радиолошким истраживањима. Циљ таквих истраживања је побољшање рада радиолога и боља брига о пацијентима.

# Референце

[1]  Brian Ripley, страна 354, [Pattern Recognition and Neural Networks](https://amzn.to/2Y1s76G), 1996

[2] Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks. Adv Neural Inf Process Syst 25. Available online at: [https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf. Accessed 22 Jan 2018](https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.%20Accessed%2022%20Jan%202018)

[3] Josh Patterson and Adam Gibson, Deep Learning: A Practitioner's Approach 1st Edition (2017)

[4] Предраг Јаничић и Младен Николић, Вештачка интелигенција (2021)

[5] Предраг Јаничић, Математичка логика у рачунарству (2009)

# [6] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do & Kaori Togashi, Convolutional neural networks: an overview and application in radiology, Papper (2018)

[7] [Shelly Soffer](https://pubs.rsna.org/author/Soffer%2C+Shelly), [Avi Ben-Cohen](https://pubs.rsna.org/author/Ben-Cohen%2C+Avi), [Orit Shimon](https://pubs.rsna.org/author/Shimon%2C+Orit), [Michal Marianne Amitai](https://pubs.rsna.org/author/Amitai%2C+Michal+Marianne), [Hayit Greenspan](https://pubs.rsna.org/author/Greenspan%2C+Hayit), Convolutional Neural Networks for Radiologic Images: A Radiologist’s Guide (Jan 29 2019)

# [8] François Chollet, Deep Learning with Python, Second Edition, (October 2021)

## [9] [Stuart Russell](http://www.cs.berkeley.edu/~russell) and [Peter Norvig](http://www.norvig.com/) Artificial Intelligence: A Modern Approach, 4th US ed.

# [10] Zsolt Nagy, Artificial Intelligence and Machine Learning Fundamentals: Develop real-world applications powered by the latest AI advances

[11] <https://keras.io/api/callbacks/>

[12] <https://keras.io/api/applications/resnet/#resnet152v2-function>

[13] <https://image-net.org/download.php>

[14] <https://www.tensorflow.org/tensorboard/get_started>

[15] <https://www.tensorflow.org/tensorboard/image_summaries>

[16] https://pandas.pydata.org/docs/

[17] Dataset <https://data.mendeley.com/datasets/rscbjbr9sj/2>