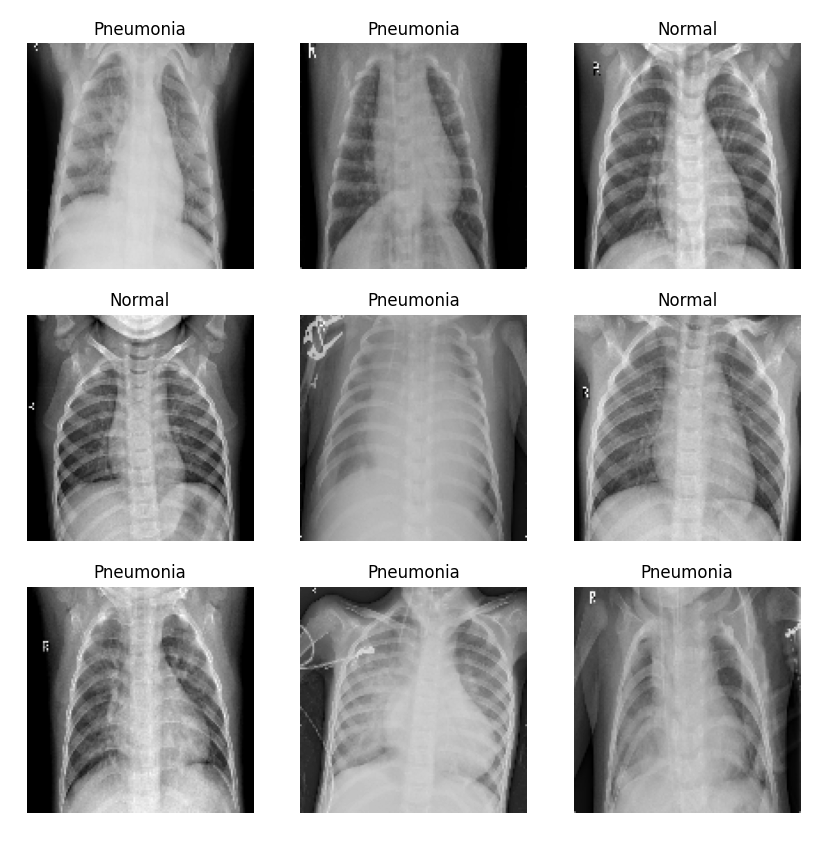
// с новите train/validation/test сетове

**Експлораторен анализ на използвания сет от данни**

За целите на дипломната работа е използван сет от **рентгенови снимки** на бели дробове. Изображенията са изтеглени от хранилището на известния в data science и machine learning среди сайт kaggle.com. В масива от данни се съдържат 5856 изображения- една част от тях на здрави бели дробове, а другата на такива с болест пневмония. Всички снимки са в JPEG формат и с различни резолюции. За анализ и обработка на дейтасета е използван програмният език Пайтън (на англ. Python). Пайтън скрипт за визуализация на част от изображенията:

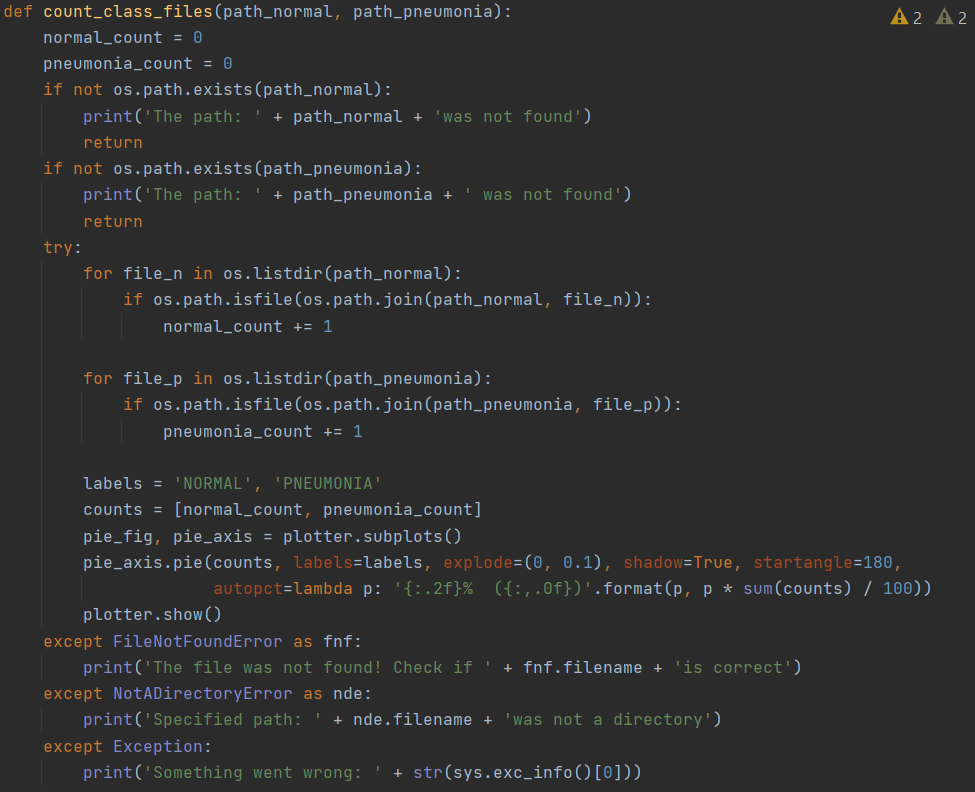




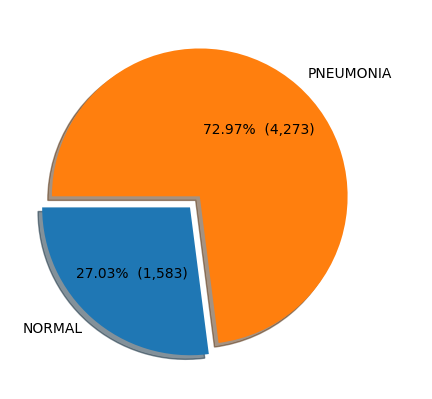
Преди да бъде използван сета за обучаване на невронните мрежи е необходимо да се направи предварителен преглед на съдържанието му. Изображенията са разделени в 2 папки :



Папка **NORMAL** съдържа набора от снимки на бели дробове без открита пневмония, а папка **PNEUMONIA** съдържа снимките с открита такава. В конкретния случай изображенията ще бъдат използвани за обучаване на невронна мрежа за бинарна класификация- клас normal и клас pneumonia. За да бъде обучението ефективно и автентично, броят на представителите на двата класа трябва да бъде приблизително еднакъв- сета от данни трябва да бъде „балансиран“. Броят и визуализацията ще бъде проверен чрез следния Пайтън скрипт:



Резултат от скрипта с подадени параметри пътищата към двете папки с изображения:



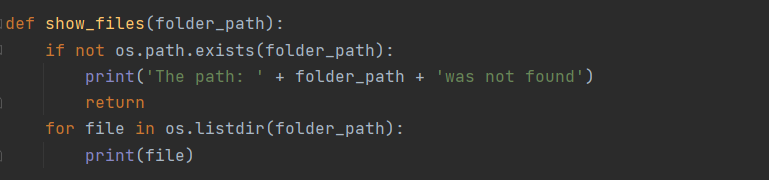
Пай диаграмата показва, че сета е силно дисбалансиран- близо 73%(4273) от всички изображения принадлежат на клас pneumonia, докато към клас normal се отнасят около 27%(1583) от снимките. Този дисбаланс представлява реален проблем- невронната мрежа може да се пренагоди в обучението си спрямо класът с повече представители- в този случай pneumonia. Така тя се „приспособява“ неправилно и е много вероятно да не класифицира успешно снимки на бели дробове, в които отсъства пневмония.

Небалансираният сет може да доведе и до проблеми при оценката на тренирания модел. При машинно обучение с такъв набор от данни съществува понятието **„парадокс на точността - the accuracy paradox“**[1]. В ситуация на съотношение на представителите на два класа 99% - 1%, в процеса на класификация, използваният алгоритъм(било то логистична регресия, SVM, невронна мрежа или др.) ще достигне над 90% точност, присвоявайки голямата част от входните данни към по- силно изразения клас. На теория, моделът се справя отлично, но на практика, той няма да генерализира добре. Така оценката на обучителния процес на база точност ще бъде изключително некоректна и няма да дава реална представа за качеството. Това явление е силно изразено при сетовете с недостатъчно количество данни - например с общ брой от 300 представители. При такъв случай, алгоритъмът няма възможност да обхване вариативността и разнообразието на класа с по-малко данни.

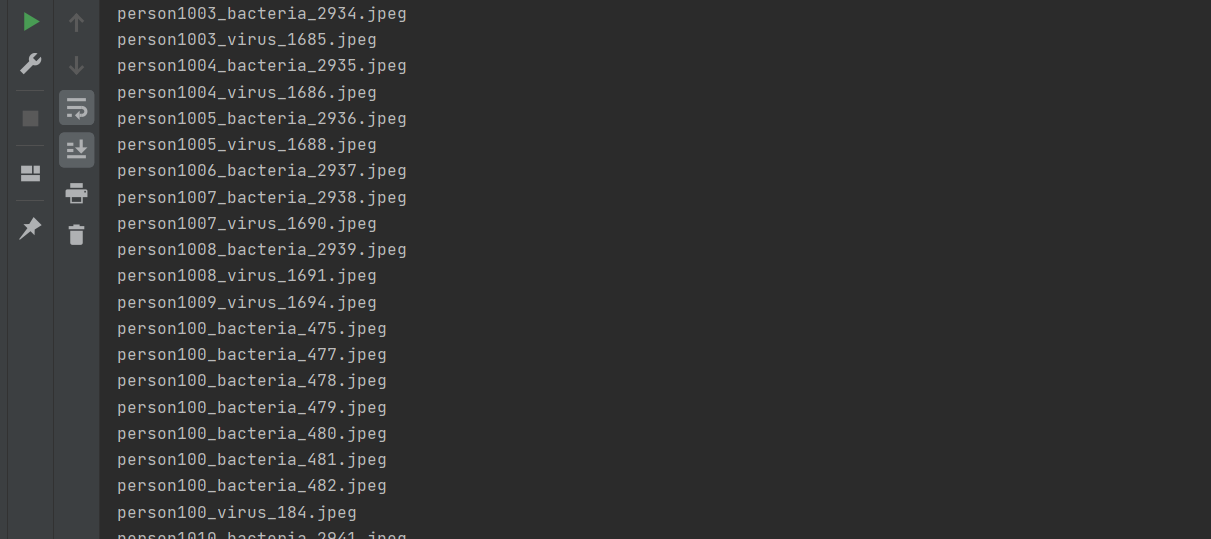
Съобразно гореописаното, трябва да бъдат взети мерки за справяне с този проблем. Съществуват разнообразни подходи и техники за справяне с небалансиран сет. Някои от най-популярните са :

* **Over-sampling –** тази техника се състои в увеличаване количеството на данните в по-слабо изразения клас. Това може да стане по различни начини:
* Чрез произволна дубликация на част от вече съществуващите данни- това е един от най-старите методи, считащ се за сравнително устойчив
* Чрез синтетично добавяне **Synthetic Minority Over-sampling Technique(SMOTE)** - една от най- разпространените техники. Генерират се изкуствени данни чрез избиране на произволен представител на класа и определяне на негови K на брой съседи в пространството на характерните за категорията черти(feature space). Създава се вектор с разстоянията до тези най-близки съседи, след което новият „синтетичен“ представител се генерира чрез умножаване на вектора с произволен коефициент в интервала (0;1) .
* Чрез адаптивно синтетично добавяне **Adaptive Synthetic Sampling(ADASYN) –** при този метод отново се добавят изкуствени представители към по- слабо изразените класове. Разликата тук е, че се използват тегла, изразяващи трудността на научаване на конкретна категория. Така се генерират изкуствени представители от тези класове, които са най- трудни за научаване.
* **Under-sampling –** При тази техника се намалява броят на представителите от по- силно изразеният клас. Съществуват различни разновидности на under-sampling
* Чрез произволно премахване на част от данните - на случаен принцип някои от представителите се изваждат от дейтасета докато се постигне желаното съотношение между класовете.
* Чрез клъстерно-базиран анализ - при този метод данните от по-ясно изразеният клас се клъстеризират с избран алгоритъм- например K-means. Така се формират групи от данни със сходни характеристики. От всеки клъстер се оставят само най-репрезентиращите клъстера данни- най-близките до центроида, а най-далечните се премахват.
* **Прилагане на теглови коефициенти** – при използването на този метод, от дейтасета не се премахват/добавят данни, а по време на обучението на всеки от класовете се присвоява тегло, което участва в крайното сформиране на вероятност за принадлежност към конкретна категория.
* **Събиране на още данни** - Това е един от тривиалните методи. Натрупването на още данни е възможно чрез проверка на различни информационни източници- хранилища, популярни платформи, учени от data science областта и дори представители от сферата на изучавания проблем. Въпреки своята тривиалност, този метод е доста труден, защото често изисква повече време, усилия и не винаги води до успех. В много от случаите, данните са регулирани чрез законова мярка и тяхното придобиване и използване се оказва нелесна задача.

За разглежданият в дипломната работа проблем ще бъде използван **Under-sampling** метода чрез премахване представители от по-ясно изразеният клас PNEUMONIA. Следва проверка на изображенията, прилежащи към този клас. Пайтън скрипт:



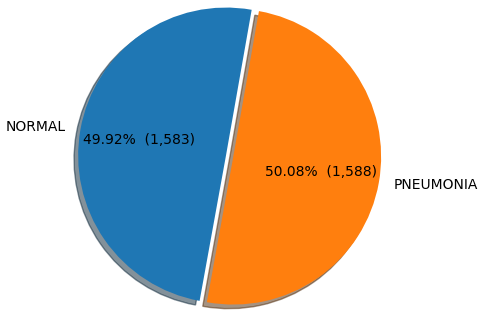
Резултат:



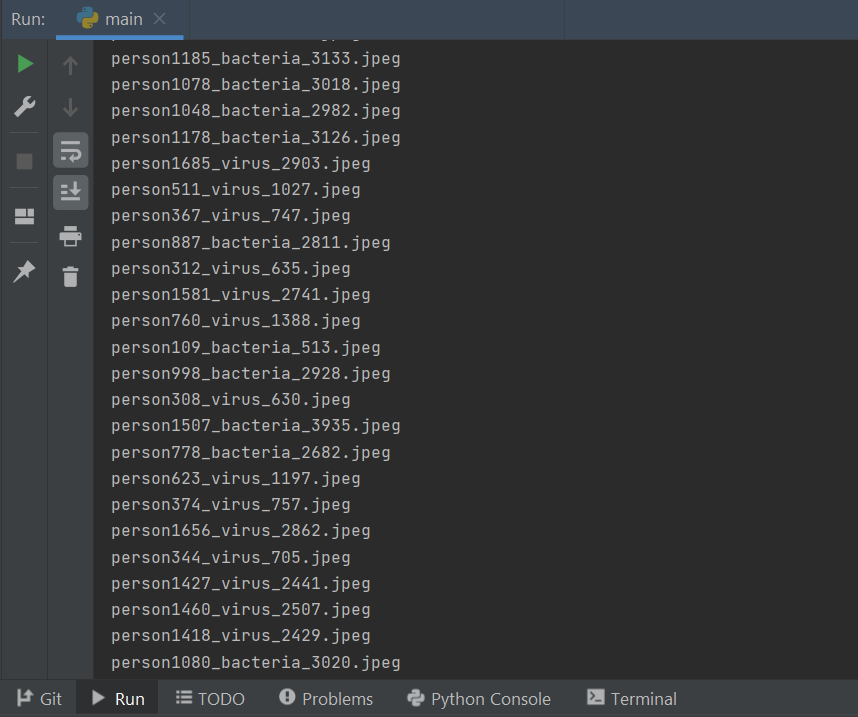
От изхода на конзолата се вижда още един потенциален проблем- **сета съдържа по няколко снимки** на едни и същи хора. Това е изключително срещано предизвикателство при обработката на медицински данни, в случая медицински изображения. В много от случаите се проследява състоянието на даден пациент в различни етапи от заболяването- правят се съответните изследвания- кръвни, томография, стрес тест, рентгенови снимки и др. Именно затова сета съдържа по няколко снимки на даден човек. Някои пациентите имат над 5-6 снимки, включени в набора от данни. Това, обаче, може да навреди на обучителния процес – невронната мрежа може да се приспособи прекалено много към характерни за някои хора черти, виждайки ги неколкократно.

С оглед на горе-описаното, приложената under-sampling техниката **няма да бъде напълно произволна.** Ръчно ще бъдат премахнати тези изображения, които принадлежат на един и същи човек, като на пациент ще бъдат оставяни в сета предимно по 2 броя рентгенови снимки, в случай на недостиг - 3. Този подход представлява комбиниран начин за **едновременно справяне** с двата проблема- както дисбалансът в дейтасета, така и наличието на няколко рентгенови снимки на един и същ човек.

След извършения ръчен under-sampling и изпълняване на Python скрипта с подаден пътя към новия набор от данни, съотношението между представителите на двата класа става:



И резултатът от прегледа на файловете :



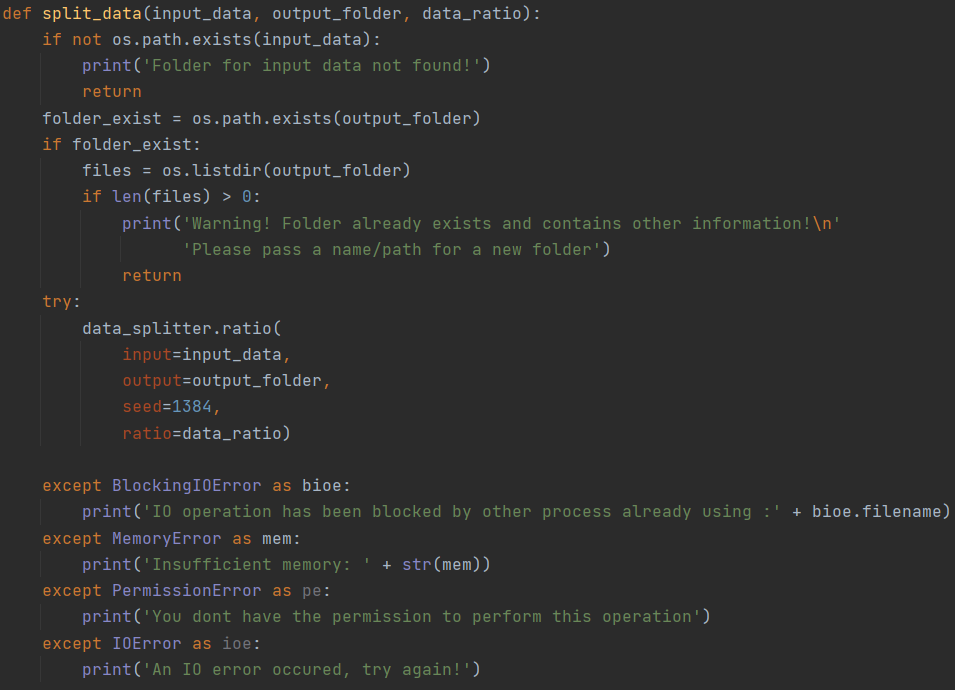
Както се вижда от пай диаграмата, съотношението между рентгеновите снимки от двата класа вече е приблизително равно- 1588 представителя на клас pneumonia, 1583 на normal. Резултатът от прегледа на файловете показва понижения брой изображения на едни и същи хора в тази категория. Недостатък на използваният under-sampling е понижаването на общият брой изображения за обучение – от 5856 на 3171. Въпреки това, смятам, че използването му е подходящо в случая, защото така едновременно се разрешават вече споменатите 2 проблема, което е предпоставка за пълноценно обучаване.

**Подготовка на дейтасета за обучаване на НМ**

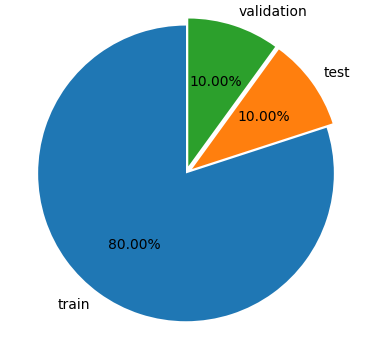
След като масивът от данни е балансиран, следва да се премине към следващата важна стъпка преди същинският процес на трениране- **разделянето на сета.** За да бъдат трениращият процес и последващата го оценка коректни, следва данните да се разделят на 3 части :

1. **Данни за трениране** – върху тях ще бъде обучавана невронната мрежа- итеративни промени в теглата, за да се открие оптималната комбинация, свеждаща грешките до минимум. В този контекст възникват и понятията „тренираща точност“, тренираща loss и други оценъчни данни. Те изразяват колко добре се справя мрежата в процеса на обучение с вече известни, предварително видени данни- т.е. служат за оценка на обучителния процес, но не дават реална представа за това колко добра генерализация се постига.
2. **Данни за валидация** – след всяка епоха от обучаването върху данните за трениране се извършва т.нар. валидация- класифициране на валидиращия сет, съдържащ невиждани досега от мрежата представители на класовете. Някои учени съотнася данните за валидация към данните за трениране, защото на база резултатите върху валидационния сет се правят настройките на хиперпараметрите на мрежата. Тук възникват понятията „валидационна точност“ и валидационна loss, даващи реална представа за това колко добре генерализира моделът. В най-добрият случай, валидационните и трениращите резултати са близки по стойност. Валидационният сет също се използва за откриване на настъпващ овърфитинг.
3. **Данни за тестване** – още наричани holdout сет. Представляват частта от данни, използвана за финална оценка на избрания вариант на модела. Тази финална оценка в повечето случаи е еднократна, но за целите на дипломната работа ще бъде извършена поотделно за най-добрият модел от съответният опит.

Разделянето на сета се извършва със следният Пайтън скрипт:



Както се вижда от скрипта, един от подаваните параметри представлява пропорциите на разделяне на сета на тренираща/валидационна/тестова част. Някои от най-честите комбинации са 60/20/20 %, 70/15/15 %, 80/10/10 %. В този случай за целите на дипломната работа е използвана конфигурацията 80% данни за трениране, 10% данни за тестване, 10% данни за валидация:



Така тренирането се осъществява с 2536 изображения, валидацията с 316, а тестването с 319.

**Опит 1.**

В този опит се използва техниката transfer learning. (Обяснения за TL и finetuning)

Архитектурата VGG16, като не са включени оригиналните крайни „top“ слоеве, а на тяхно място са поставени подходящи за текущия разглеждания проблем други такива. Като стойности на теглата са заредени получените при трениране върху imagenet сета за данни, комбиниращ… Код на архитектурата:

base\_mdl = krs.applications.vgg16.VGG16(include\_top=False, input\_shape=input\_shape, weights='imagenet')  
base\_mdl.trainable = False  
inputs = krs.Input(input\_shape)  
inp = data\_augm(inputs)  
inp = krs.applications.vgg16.preprocess\_input(inp)  
val = base\_mdl(inp, training=False)  
val = krs.layers.Flatten()(val)  
val = krs.layers.Dense(2048, activation="relu")(val)  
outputs = krs.layers.Dense(1, activation='sigmoid')(val)  
return krs.Model(inputs, outputs)

За да се намали пренагаждането на модела към тренировъчните данни и съответно той да бъде по- устойчив и да се справя добре с невиждани досега изображения, към тренировъчният pipeline е включен и механизмът за обогатяване на данните(data augmentation). Използван е 1 плътно свързан Dense слой с 2048 неврона и relu активираща функция. Като последен слой е поставен същинският класификатор със сигмоидна активационна функция.

**Тест 1.**

Параметри на теста:

Folder: vgg\_2

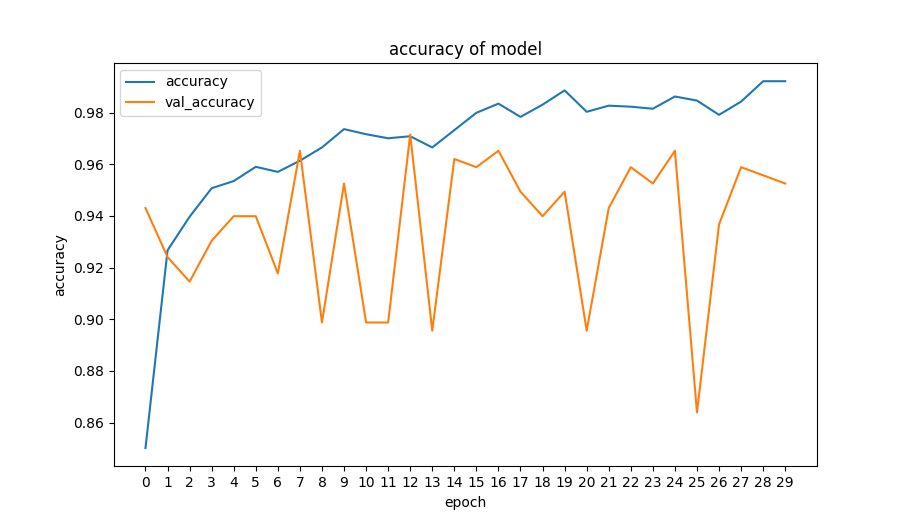
Image size: 128x128x3

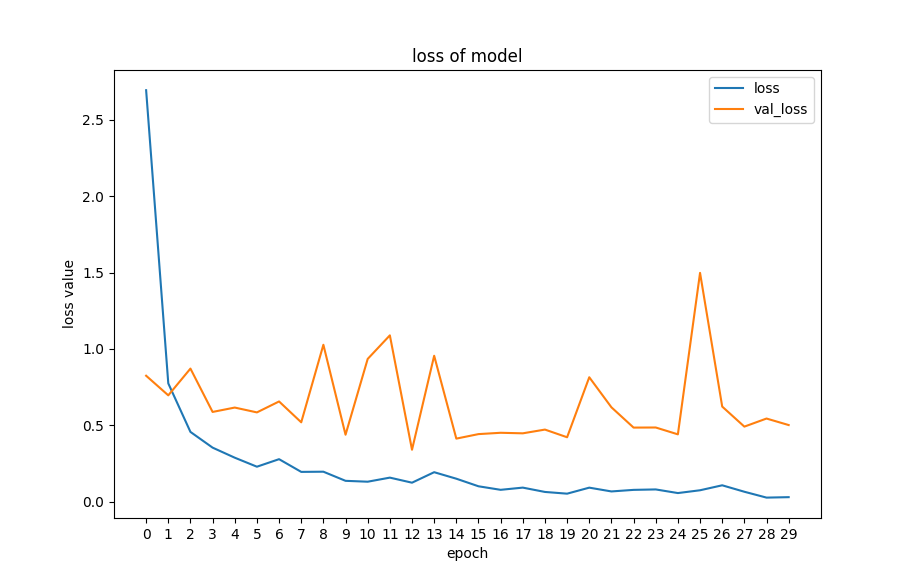
Batch size: 32

Epochs: 30

model.compile(  
 optimizer=keras.optimizers.Adam(learning\_rate=0.001),  
 loss="binary\_crossentropy",  
 metrics=["accuracy", "AUC",  
 keras.metrics.TruePositives(),  
 keras.metrics.FalsePositives(),  
 keras.metrics.TrueNegatives(),  
 keras.metrics.FalseNegatives()])

Резултати от тестването:





От графиките се вижда, че мрежата се учи успешно върху данните за трениране- точността при трениране нараства, а loss стойността- намалява. В стойностите при валидацията, обаче се наблюдава осцилиране на графиката. Например за стойностите на валидационната точност - те варират в интервала 90-96%. В контекста на трансферираното обучение битува и понятието „фина настройка“(fine tuning). Тя включва използване на ниска стойност на параметъра learning rate, защото се очаква предварително натрупаното знание (в този случай заредените тегла от imagenet дейтасета) да спомогне за бързото намиране на оптимално решение. За горния експеримент е зададен learning rate= 0.001- сравнително висока стойност, въпреки използвания адаптиращ оптимизационен метод. Това може да бъде причина за пропускане на оптималното решение(optimum overshooting), резултиращо в осцилирането на графиката.

От резултатите се забелязва, че след епоха 13, валидационната точност започва да спада, докато тази при трениране продължава да се увеличава. Подобна тенденция се забелязва и при стойността на loss функцията. Това е т.нар. пренагаждане на модела спрямо обучителния сет. Въпреки подобряващата се тренираща точност, мрежата започва да се справя по- лошо с невиждани досега данни и губи качеството си да генерализира добре. Съществуват различни техники за справяне с overfitting-a. Една от тях е въвеждането на dropout слой, който изключва част от невроните при обучението(с определена вероятност). На база резултатите и разсъжденията върху тях, следва да бъдат направени следните промени към архитектурата/обучителния процес:

* Намаляване на learning rate параметъра
* Добавяне на още един плътносвързан слой
* Добавяне на Dropout слой

**Тест 2.**

В този тест са отразени промените в архитектурата, изяснени по-горе:

base\_mdl = krs.applications.vgg16.VGG16(include\_top=False, input\_shape=input\_shape, weights='imagenet')  
base\_mdl.trainable = False  
inputs = krs.Input(input\_shape)  
inp = data\_augm(inputs)  
inp = krs.applications.vgg16.preprocess\_input(inp)  
val = base\_mdl(inp, training=False)  
val = krs.layers.Flatten()(val)  
val = krs.layers.Dense(2048, activation="relu")(val)  
val = krs.layers.Dropout(0.1, seed=1333)(val)  
val = krs.layers.Dense(2048, activation="relu")(val)  
outputs = krs.layers.Dense(1, activation='sigmoid')(val)  
return krs.Model(inputs, outputs)

Добавен е втори Dense слой с 2048 неврона и ReLu активационна функция. Между двата плътносвързани слоя е добавен Dropout слой с вероятност за включване- 0.1.

Освен промените в архитектурата, използвана е и по-ниска стойност на параметъра learning rate= 0.00001. Така параметрите за теста са следните:

Folder: vgg\_4

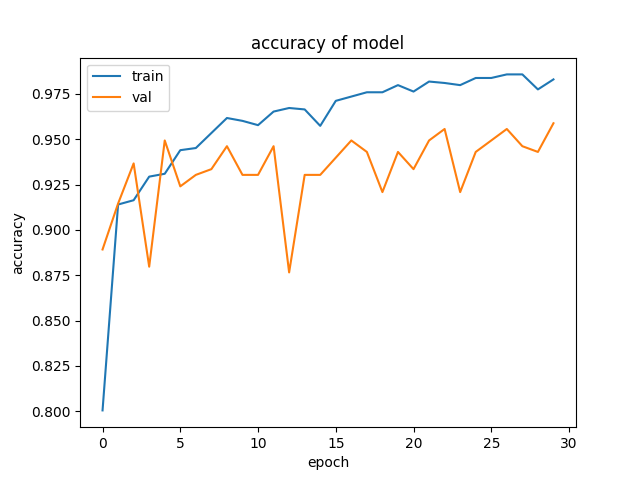
Image size: 128x128x3

Batch size: 32

Epochs: 30

Learning rate: 0.00001 (1.10-5)

Резултати:





Забелязва се известно подобрение спрямо предните резултати. Пренагаждането на модела е намалено- в графиката за точността се наблюдава лека тенденция за повишаване по време на валидация, но все пак след епоха 8 се запазва разлика между валидационна/тренираща точност от порядъка на 2-3%. В графиките на ценовата функция, обаче не личи значим прогрес след епоха 5 за валидационната точност, докато мрежата успешно се обучава и достига стойности от 0.06 при трениране. Необходим е още един тест с по- ниска стойност на параметъра learning rate.

Интерес буди и фактът, че в началните епохи, точността при валидация е по-висока от тази при трениране. Това се случва поради две причини:

**1. Data augmentation/dropout**- Тези техники за намаляване на овърфитинга са активни само по време на тренирането(но не и по време на валидацията). Тоест, невронната мрежа изкуствено е затруднена при обучението, което води и до по-лоши стойности на точността. Въпреки това, тя ще генерализира по-добре.

**2. Оценката на резултатите при трениране/валидация**- към момента, реализацията на tensorflow(в частност keras) оценява резултатите при трениране и валидация по различен начин. При трениране, резултатите се изчисляват per-batch- в конкретния случай на всеки 32 изображения, защото batch size=32. След това крайният резултат се получава чрез средно аритметично на стойностите за отделните batch-ове. В първите от тях, мрежата не се справя добре и това води до ниска batch accuracy и респективно висока batch loss. След усредняване на стойността на база всички batch-ове от епохата, то лошите стойности притеглят средната стойност надолу. Друг е начина на изчисление при валидацията- използват се крайните тегла(в случая най-добрите), получени в епохата след трениране, което води до по-добри стойности на валидацията в началото на обучителния процес.

**Тест 3.**

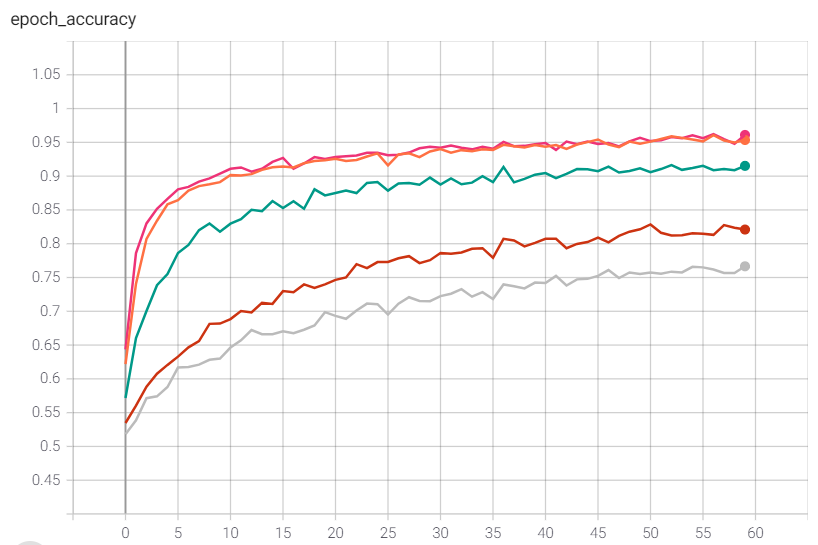
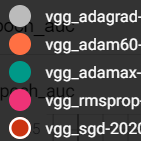
В предните 2 теста е използван един от най – разпространените и съвременни оптимизационни алгоритми- Adam(adaptive optimizer), доказано даващ много добри резултати при трениране на невронни мрежи. Съществуват обаче и други видове оптимизационни алгоритми. Подбирането на оптимизатор е важна част от обучителният процес. Затова в този тест ще бъде проверено кой оптимизатор е най-подходящ за текущия набор от данни и респективно конкретната НМ. Използваната архитектура е като тази в тест 2, а използваният Learning Rate е 1.10-6.

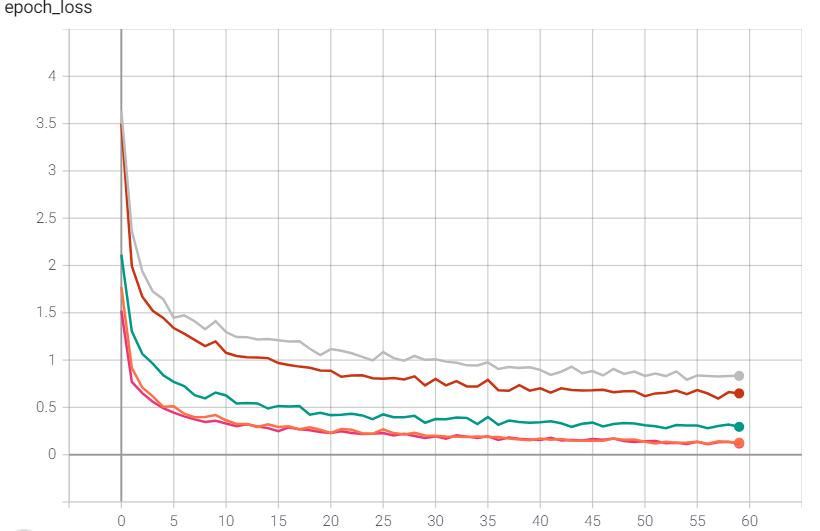
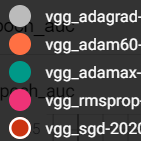
Тествани оптимизационни методи:

* Adam
* RMSProp
* Adagrad
* Stochastic Gradient Descent
* Adamax

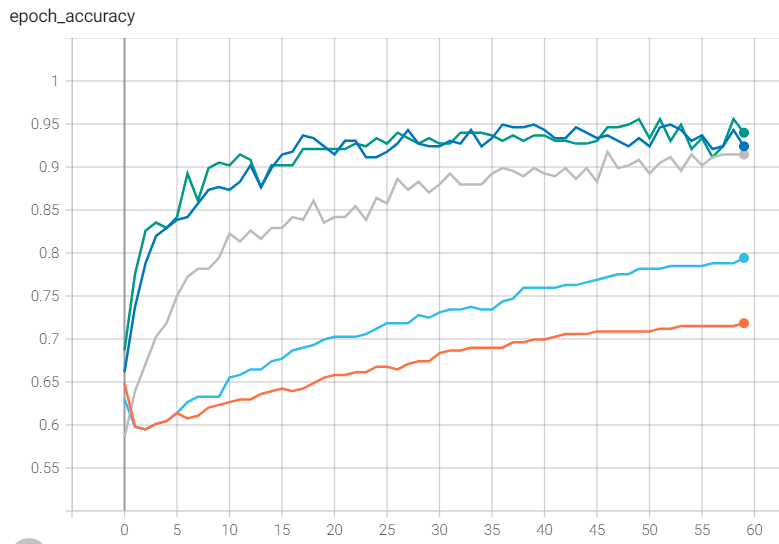
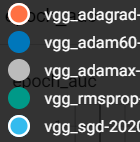
След трениране за 60 епохи се получават следните резултати:

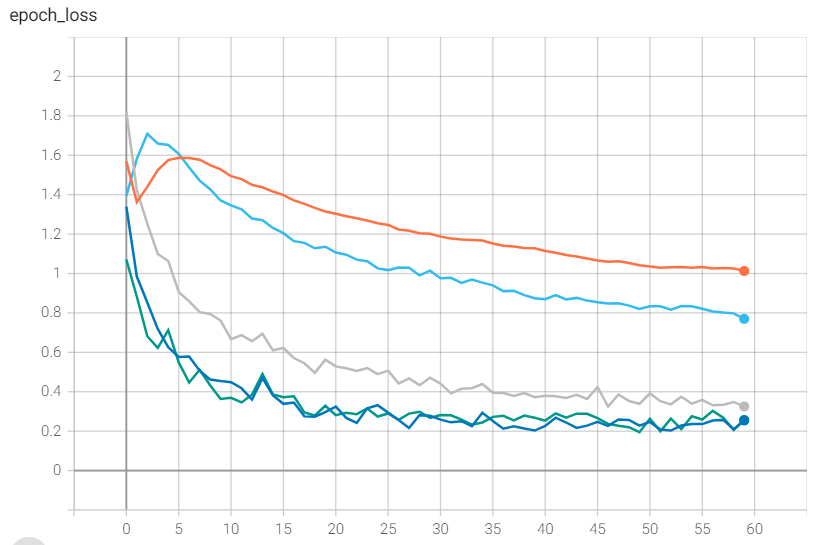
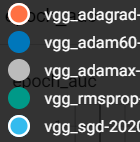
За трениращите точности и loss функция:





За валидационните точност и loss функция:





Както се вижда от графиките, най- зле се представят SGD и Adagrad методите, а най- добре- RMSProp и Adam. Обучението с Adam и RMSProp е чувствително по-бързо, като около 25-та епоха стойността на loss функцията е сведена до 0,2. Друга особеност, която прави впечатление в графиките е повишеното осцилиране при някои от методите- Adamax, RMSProp и Adam- това се дължи на динамичното адаптиране на learning rate(LR) в процеса на обучение, защото тези оптимизатори са адаптивни(за разлика от SGD например). Забелязва се още, че осцилирането е по- силно в началните епохи- това е така, защото тогава са по-големи промените в LR и съответно в стойностите за теглата.

Разбира се, всеки оптимизационен алгоритъм изисква различно настройване на хиперпараметрите си, но може да се обобщи, че за текущият разглеждан проблем Adam и RMSProp изискват минимални допълнителни настройки и достигат сравнително бързо оптималното или близко до оптималното решение. Кривата на loss функцията за SGD намалява много плавно, което се дължи на ниската стойност на learning rate параметъра. SGD най - често се използва с по-висока стойност на LR и поетапното му намаляване. Това става обикновено под формата на LearningRateScheduler, намаляващ LR по предварително дефинирана от разработчика схема. Предимствата на Adam оптимизатора са :

* Лесен за използване
* Не се нуждае от много настройки
* Адаптира самостоятелно Learning Rate параметъра и поддържа различни LR за различните параметри
* Изисква по-малко памет- спрямо по- старите методи като SGD
* Подходящ за градиенти с високо ниво на шум
* Изчислително ефикасен
* Бърз

На база горните графики за следващите тестове отново ще бъде използван Adam оптимизатора, предвид добрите резултати, които се постигат и множеството му предимства.

**Тест 4.**

Тук ще бъде използвана същата архитектура, както в тест 2, но скоростта на обучение ще бъде намалена на 1. 10-6. Очакваният резултат е намаляване на осцилирането. Параметри на тестването:

Folder: vgg\_5

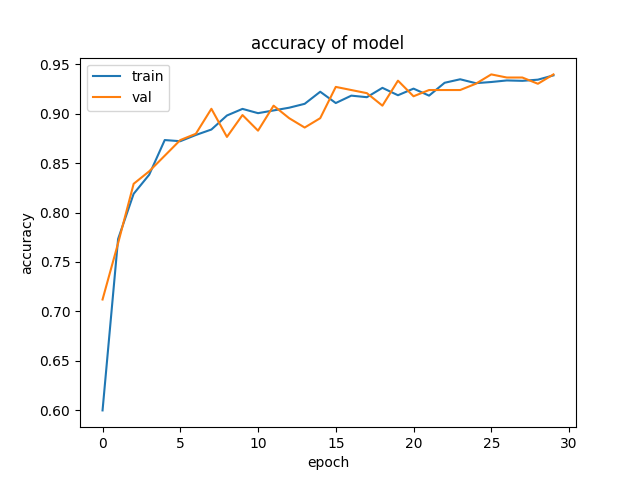
Image size: 128x128x3

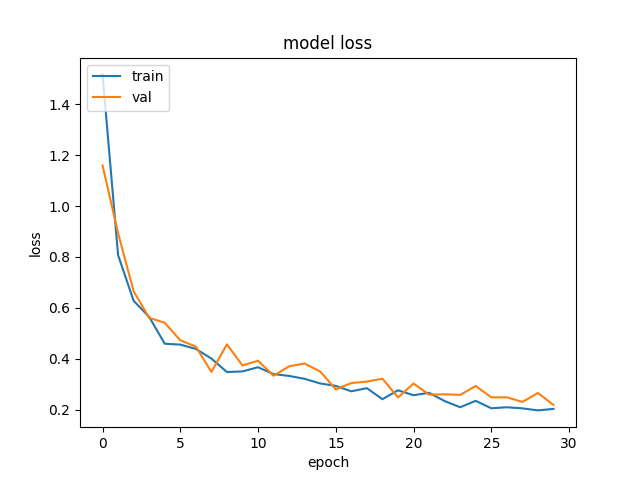
Batch size: 32

Epochs: 30

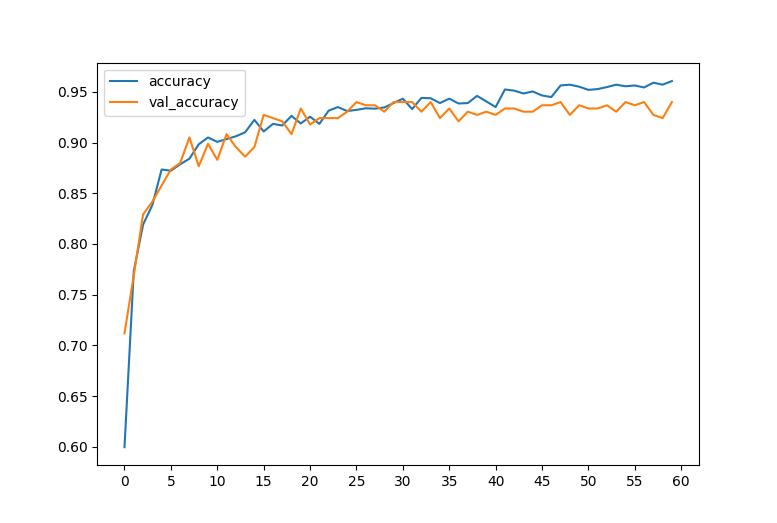
Learning rate: 0.000001 (1.10-6)

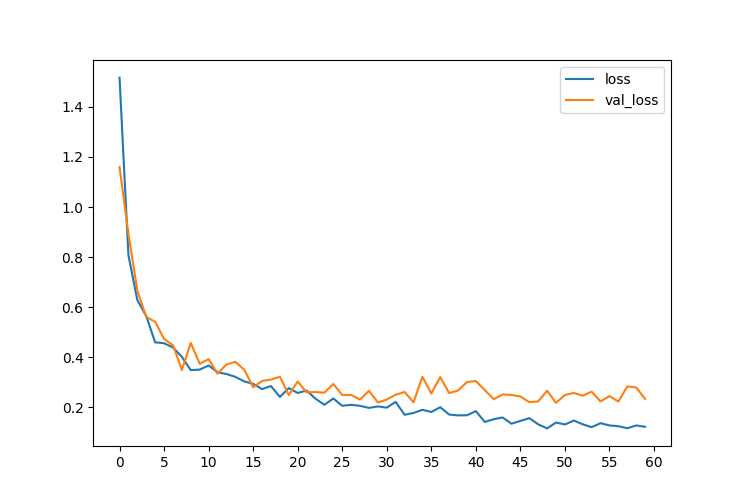
Резултати:





От графиките се вижда намаляване на осцилирането и стабилизиране на кривите. В продължение на обучението се вижда едновременно подобряване както на валидационните, така и на трениращите резултати. Стойностите при валидация/трениране са много близки - кривите са почти една върху друга. Максималната достигната точност е около 93%. Може да се каже, че мрежата генерализира добре и дори позволява още обучение, тъй като не се наблюдава спад/стагнация при валидирането. Следва трениране за още 30 епохи:





Вижда се, че оптималната достигната точност е 93,98% в епоха 34. В следващите епохи, валидационните резултати започват да се различават от тези при трениране и съответно модела започва да се пренагажда. Зоните, където се случва това са оградени със зелен цвят. По-нататъшно трениране на **тази архитектура** на НМ с **тези стойности** на хиперпараметрите няма да доведе до по-добри резултати, а напротив- мрежата започва да разчита прекалено много на train сета и няма да се справи добре с невиждани досега данни(което личи и от валидационната крива след епоха 35).

Тенденцията за подобряване на резултатите при трениране навеждат на мисълта, че мрежата „има какво да научи още“- кривите на точността и loss функцията се подобряват при трениране, вместо да се наблюдава насищане/спад. В този случай единственото ограничение, налагащо спиране на обучаващия процес е overfitting-ът на модела. Възможни са два подхода, за да се повиши точността при валидация и да се извлече маскимума от данните, без да се наруши генерализацията. Тези два подхода са- опростяване на архитектурата и прилагане на регуляризация.

**Подход 1 - Опростяване на архитектурата.**

Възможна причина за пренагаждане на модела е прекалено сложната архитектура на невронната мрежа. В оригиналната VGG16 имплементация, като top слоеве са използвани 2 плътносвързани слоя с **4096** неврона. В тренирано в този опит като крайни слоеве са използвани също 2 dense слоя, но с **2048** неврона. Следва да се провери възможно ли е намаляване/забавяне на пренагаждането напред във времето чрез редуциране броя на невроните в тези слоеве. Така архитектурата придобива следния вид:

base\_mdl = krs.applications.vgg16.VGG16(include\_top=False, input\_shape=input\_shape, weights='imagenet')  
base\_mdl.trainable = False  
inputs = krs.Input(input\_shape)  
inp = data\_augm(inputs)  
inp = krs.applications.vgg16.preprocess\_input(inp)  
val = base\_mdl(inp, training=False)  
val = krs.layers.Flatten()(val)  
val = krs.layers.Dense(1024, activation="relu")(val)  
val = krs.layers.Dropout(0.1, seed=1333)(val)  
val = krs.layers.Dense(1024, activation="relu")(val)  
outputs = krs.layers.Dense(1, activation='sigmoid')(val)  
return krs.Model(inputs, outputs)

Параметри на теста:

Folder: vgg\_6

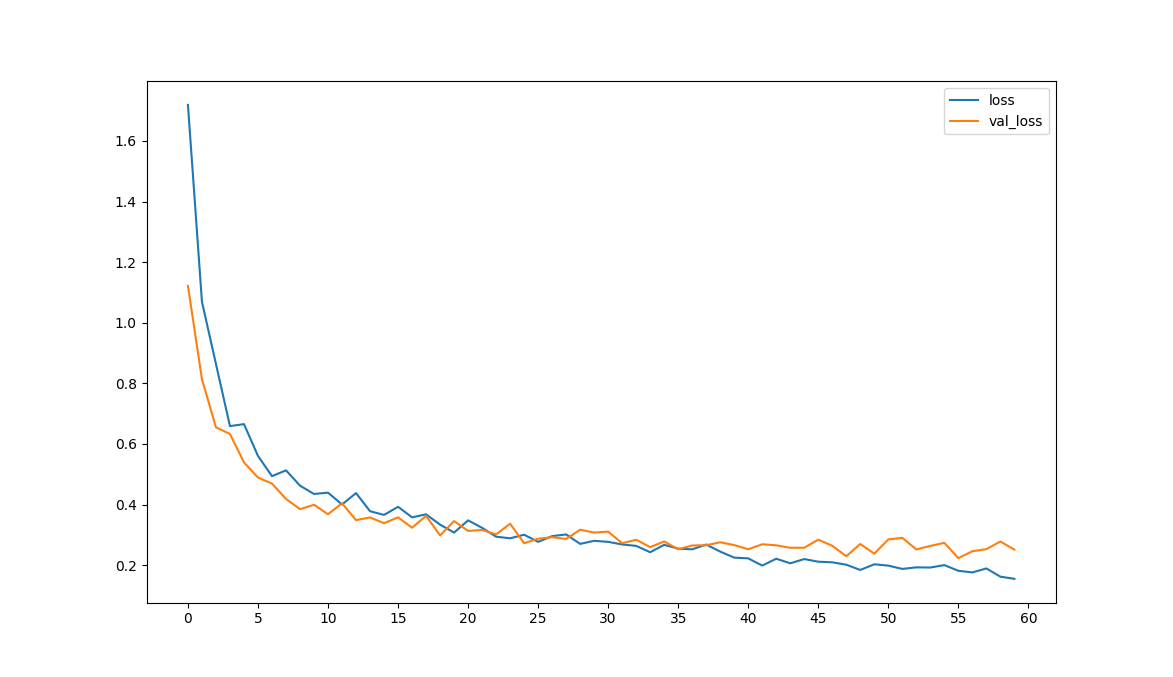
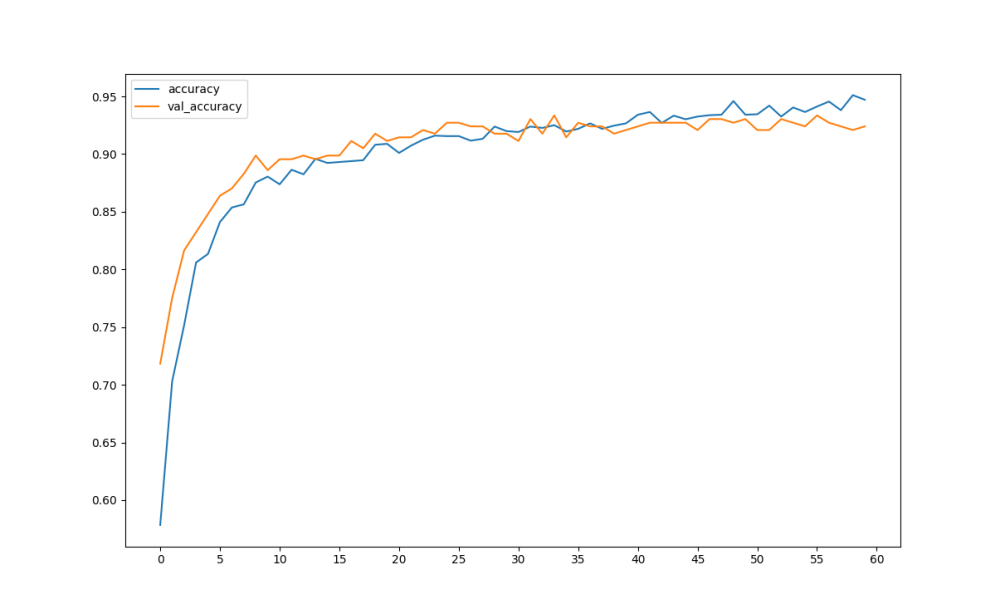
Image size: 128x128x3

Batch size: 32

Epochs: 60

Learning rate: 0.000001 (1.10-6)

Резултати:



При извършеното трениране за 60 епохи се вижда, че достигнатата максимална валидационна точност е 93,35%, а минималната стойност на ценовата функция- 0.229. След епоха 48 мрежата започва да се пренагажда спрямо трениращия сет. Може да се заключи, че направената редукция на броя неврони в top слоевете на мрежата от 2048 на 1024 не спомага намаляването/забавянето на overfitting-a, а дори максимално достигнатата валидационна точност е малко по-ниска(93,35 спрямо 93,98%). Следователно, опростяването на мрежата не носи ползи за конкретния експеримент.

max val acc- 93,35% в епоха 34

min val loss – 0.229 в епоха 48

**Подход 2 - Прилагане на регуляризация**

Регуляризацията е метод за справяне с пренагаждането на различни ML модели към трениращия сет. Тя бива два вида – L1 и L2. В естеството си, регуляризацията се състои в свиване големината на теглата на параметрите, резултиращо в намаляване на сложността на модела. L1 и L2 регуляризациите се различават по това, че L1 позволява свиване на теглата до 0(водещо до премахване на чертите, които мрежата счита за маловажни), докато L2 не позволява свиване до 0(т.е не се губи информация).

В този случай ще бъде използван по-малко рестриктивния вариант L2. Регуляризацията ще се прилага само върху изходните top слоеве(Dense с 2048 неврона), тъй като в експеримента се използва техниката transfer learning и теглата на feature extractor слоевете са замразени. Отразяване на регуляризацията в кода на архитектурата:

base\_mdl = krs.applications.vgg16.VGG16(include\_top=False, input\_shape=input\_shape, weights='imagenet')  
#base\_mdl.summary()  
base\_mdl.trainable = False  
inputs = krs.Input(input\_shape)  
inp = data\_augm(inputs)  
inp = krs.applications.vgg16.preprocess\_input(inp)  
val = base\_mdl(inp, training=False)  
#val = krs.layers.GlobalAveragePooling2D()(val)  
val = krs.layers.Flatten()(val)  
val = krs.layers.Dense(2048, activation="relu",  
 kernel\_regularizer=krs.regularizers.l2(0.000001))(val)  
val = krs.layers.Dropout(0.1, seed=1333)(val)  
val = krs.layers.Dense(2048, activation="relu",  
 kernel\_regularizer=krs.regularizers.l2(0.000001))(val)  
#val = krs.layers.Dropout(0.3)(val)  
outputs = krs.layers.Dense(1, activation='sigmoid')(val)  
return krs.Model(inputs, outputs)

Параметри на теста:

Folder: vgg\_v8reg

Image size: 128x128x3

Batch size: 32

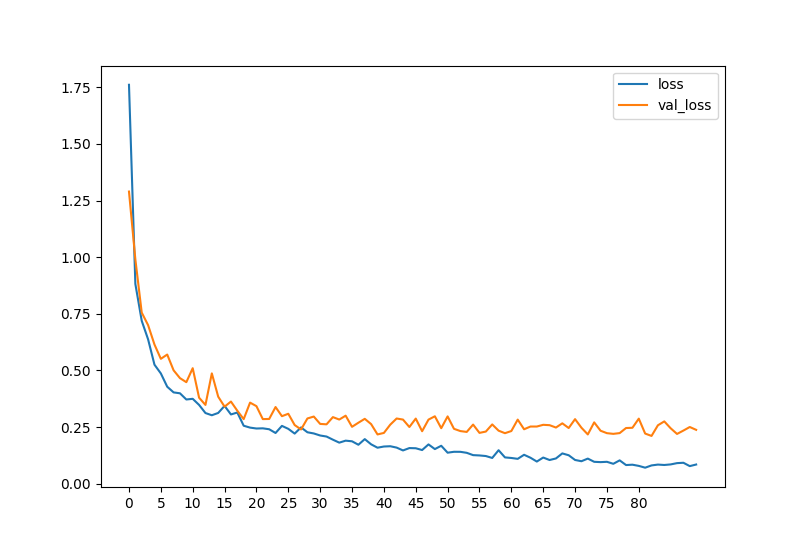
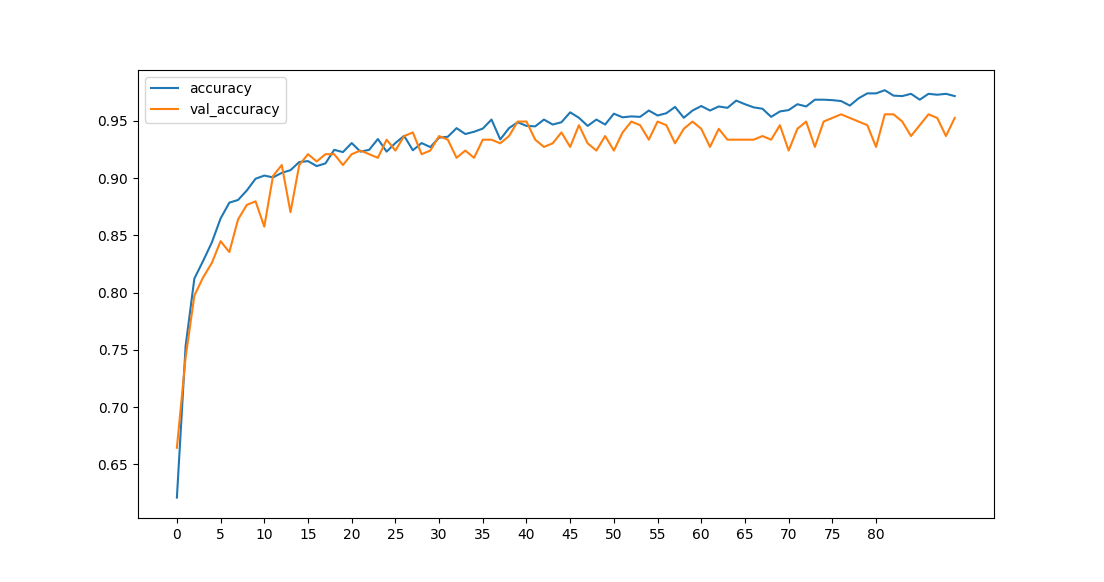
Epochs: 90

Learning rate: 0.000001 (1.10-6)

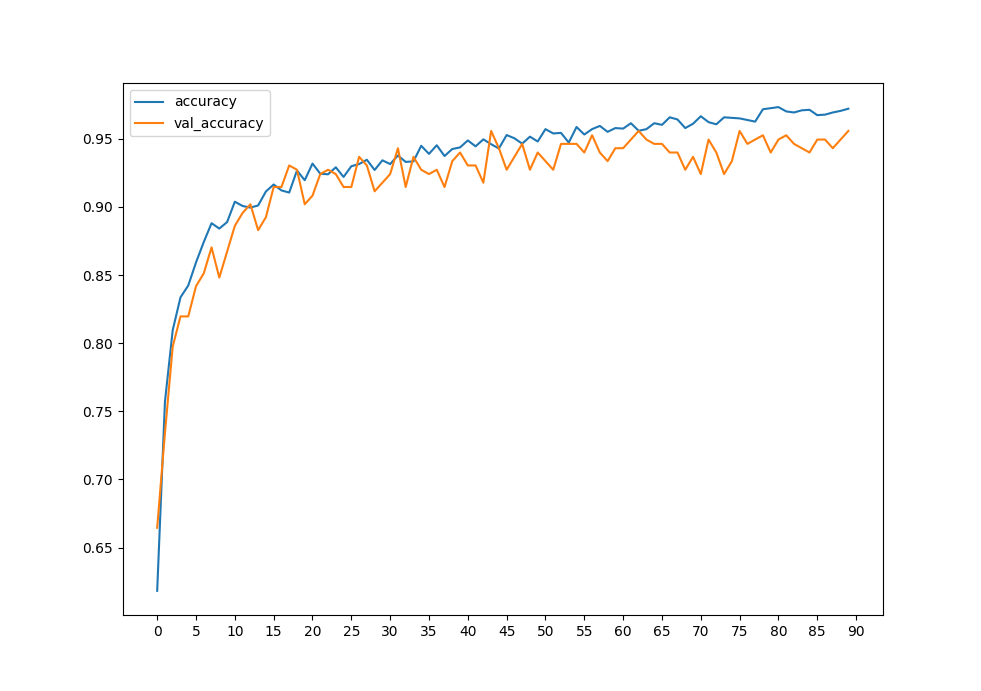
Optimizer: Adam

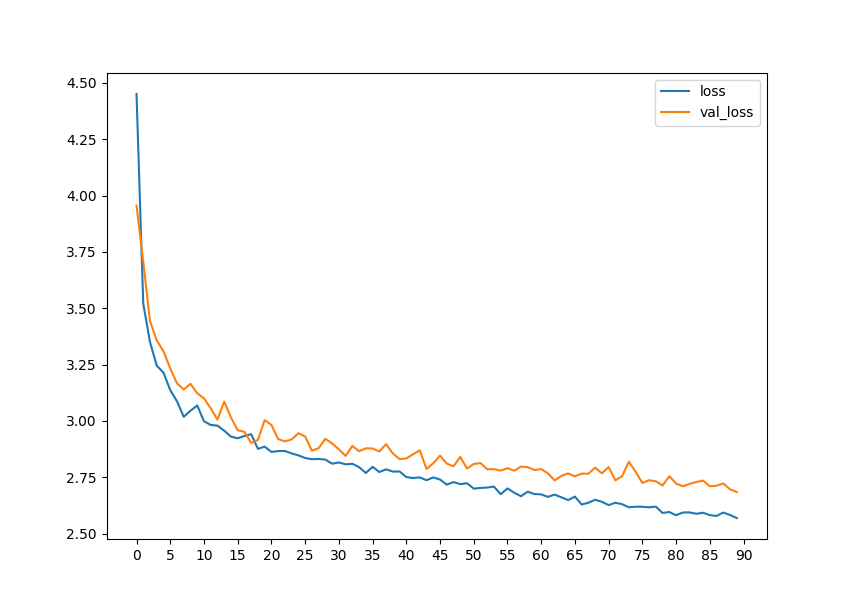
L2 regularization rate: 0.000001

Резултати:



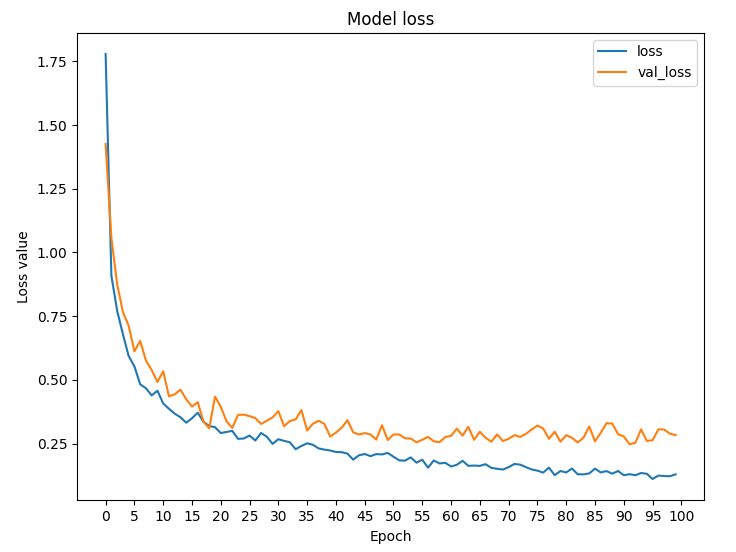
Както се вижда от графиките, максималната валидационна точност е 94,93% епоха 40. След това не личи особено подобрение в кривата на точността, а в кривата на ценовата функция се вижда стагнация/увеличаване. Сравнявайки графиките на тестване с и без регуляризация, не се забелязва голяма полза от регуляризирането. За теста бе използвана ниска стойност на L2, която видимо все още не ограничава достатъчно невронната мрежа в обучението ѝ, следователно тя трябва да бъде увеличена. Затова ще бъде променена регуляризационната стойност на 0.0005. Тази стойност е предложена и от Франсоа Шоле(създателят на Keras) за мрежата Inception, тренирана върху ImageNet дейтасета. След ново трениране за 90 епохи, този път с регуляризация 0.0005 се получават следните графики:

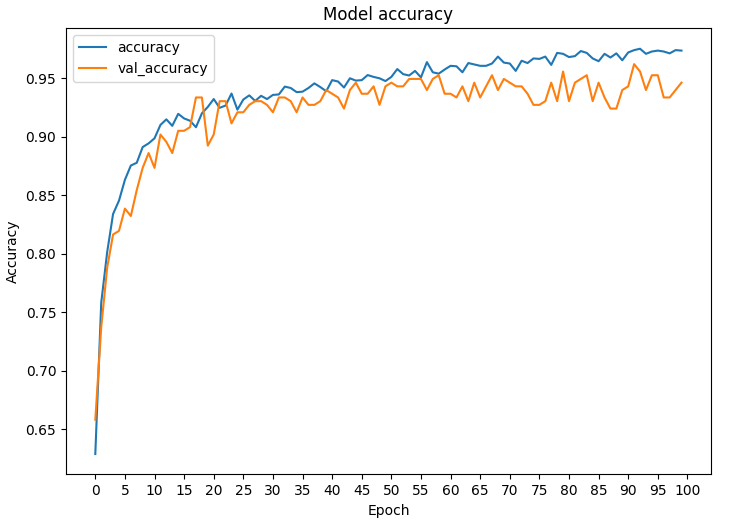




Този път се вижда по-висока достигната стойност на валидационната точност- 44 епоха – 95,56%. Тази стойност е с около 2% по-голяма, от колкото достигнатата при обучение без регуляризация. След епоха 63 се наблюдава пренагаждане на модела. Интерес буди повишената стойност на ценовата функция. В останалите тестове, още след първите няколко епохи тя е сведена под 1, като насищане се наблюдава около 0,2. В този случай обаче, тя има доста по- високи стойност- над 2,5. Това е така именно заради приложената регуляризация, непозволяваща на мрежата да наподоби прекалено много трениращия сет и да се получи овърфитинг. Съдейки по високата стойност на загубите, L2= 0.0005 за конкретната архитектура се оказва прекалено силна, затова ще бъде намалена стойността на L2=0.00001. Следва ново трениране със същата конфигурация, но обновената стойност на регуляризацията.

Резултати:





На база горните графики може да се направи извода, че стойността на регуляризация 0.00001 е подходяща в случая- достигната е валидационна точност от 95.25% в епоха 59, като стойността на загубите е 0.26, т.е. успешно е подобрен моделът без да са наложени силни рестрикции. След епоха 65 се наблюдава спад на валидационната точност със спорадични пикове(като този в епоха 92). Поради тенденцията за спад и фактът, че оптимизаторът би трябвало да е стабилизирал learning rate стойностите за параметрите(промените в теглата да са плавни), може да се заключи, че тези пикове са със стохастичен характер.

Обобщение на резултатите от регуляризацията:

|  |  |  |
| --- | --- | --- |
| Параметри    Величина | Max тренираща точност(%)  (без овърфитинг) | Max валидационна точност (%) |
| Без L2 регуляризация | 94,36 | 93,99 |
| L2= 0.000001 | 94,87 | 94,93 |
| L2=0.00001 | 95,39 | 95,25 |
| L2= 0.0005 | 95,26 | 95,96 |

Приложената регуляризация доведе до желания резултат- наблюдава се намаляване на пренагаждането, позволяващо по-продължително трениране и повишаване на валидационната точност.

Регуляризацията трябва да бъде прилагана внимателно, защото при прекалено висок коефициент, тя може да доведе до underfitting. Затова е нужно да се намери подходящата стойност, която хем да намали overfitting-a и да може да се извлече по- висока точност, хем да не доведе до underfitting и алгоритъмът да се справя зле и с трениращите данни(да се понижи точността). В този случай е достигнато желаното намаляване на овърфитинга и повишаване на точността, като най-балансиран вариант представлява обучаването с L2=0.00001. Именно затова и точно този модел ще бъде използван във финалната оценка.

**Опит 2.**

В този опит ще бъде използвана собствена имплементация на невронна мрежа, базирана на Inception модула. Идеята на този модул е да се разграничи частично от традиционните дълбоки невронни мрежи. В него се извършват паралелно конволюции с филтри с различни размери(най-често 1x1,3x3,5x5), след които резултатите се обединяват и подават на следващият такъв модул. Съществуват различни видове Inception модули, спрямо различните версии на Inception мрежата.

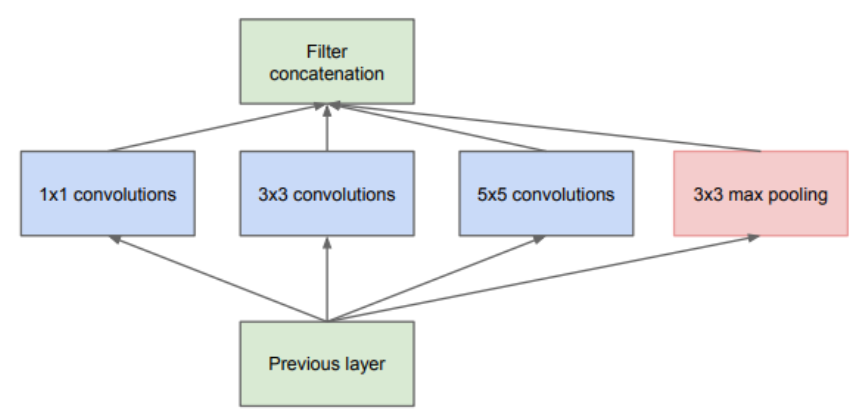
Тук ще бъде използвана традиционна версия на Inception модула, използвана в мрежата GoogLeNet, разработена от Гугъл през 2015г. Той включва:

* Конволюция с филтри 1x1 с relu активация и нулев падинг
* Конволюция с филтри 3x3 с relu активация и нулев падинг
* Конволюция с филтри 5х5 с relu активация и нулев падинг
* Макс пулинг слой

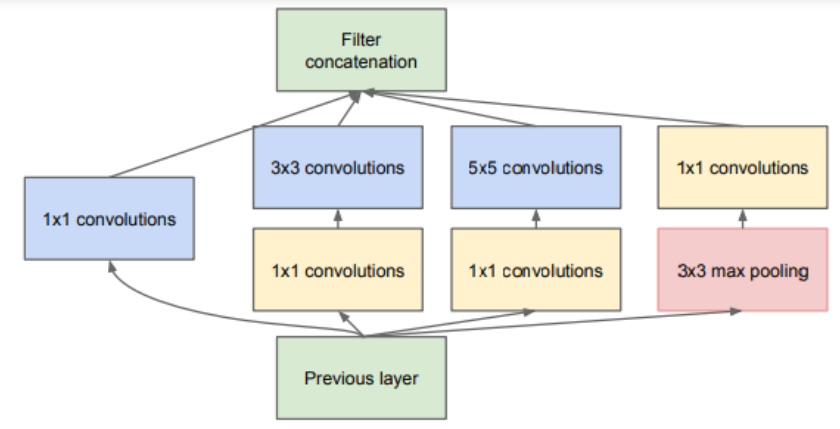
Код на Inception клетката:

def my\_inception(layer\_input, x1, x2\_reduction, x2, x3\_reduction, x3, x4):  
 # 1x1 conv  
 conv1 = krs.layers.Conv2D(x1, (1, 1), padding='same', activation='relu')(layer\_input)  
 # 3x3 conv  
 conv3 = krs.layers.Conv2D(x2\_reduction, (1, 1), padding='same', activation='relu')(layer\_input)  
 conv3 = krs.layers.Conv2D(x2, (3, 3), padding='same', activation='relu')(conv3)  
 # 5x5 conv  
 conv5 = krs.layers.Conv2D(x3\_reduction, (1, 1), padding='same', activation='relu')(layer\_input)  
 conv5 = krs.layers.Conv2D(x3, (5, 5), padding='same', activation='relu')(conv5)  
 # 3x3 max pooling  
 pool = krs.layers.MaxPooling2D((3, 3), strides=(1, 1), padding='same')(layer\_input)  
 pool = krs.layers.Conv2D(x4, (1, 1), padding='same', activation='relu')(pool)  
 # concatenate filters, assumes filters/channels last  
 layer\_out = concatenate([conv1, conv3, conv5, pool], axis=-1)  
 return layer\_out

Така наречената „наивна“ имплементация прилага директно конволюция на филтрите с различните размерности в следния вид:

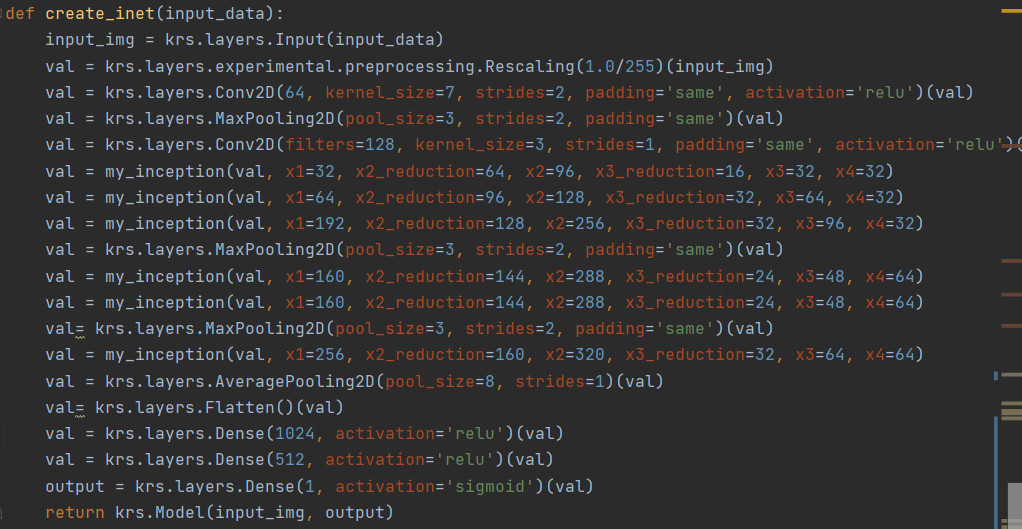


Това обаче води до необходимостта от поддържане на много голям брой параметри, водещ до по-бавно и изискващо много ресурси трениране на мрежата. Именно затова, авторите на модула предлагат вариация, използващ конволюция с филтри с размер 1х1 преди прилагането на останалите филтри. Това намалява размерността, респективно и параметрите. Именно тази вариация е използвана и в текущата имплементация:



Този модул е използван в имплементацията на мрежата, като са подавани различни стойности за броя на различните филтри. Важно е да се отбележи, че архитектурата на невронната мрежа по-долу е началната такава(т.е. subject-to-change) и върху нея ще бъдат прилагани подходящи промени на база постигнатите резултати от трениращият процес, в случай, че това бъде необходимо.

Код на архитектурата:



Както се вижда от кода, стойностите на пикселите на входните изображения са скалирани в интервала [0;1]. Преди прилагане на Inception модула са извършени 2 конволюции и MaxPooling операция за екстракция на основните черти на изображението. След приложените Inception блокове е извършен AveragePooling за намаляване на размерността и екстракция на най-важните черти. Flatten слоя извършва подготовката на данните за подаването им на плътносвързаните Dense слоеве със съответно 1024 и 512 неврона. Като изходен слой е използван Dense с 1 неврон и сигмоидна активационна функция, свиваща резултата в рамките на 0 и 1, отразяващ принадлежността към съответния клас. За разлика от Опит 1, където техниката на трансферирано обучение изискваше използване на 3-канални изображения, в този опит ще бъдат използвани изображения в режим **на сива скала с 1 канал**.

**Тест 1.**

Извършено е трениране за 50 епохи с гореописаната архитектура. Както се вижда, към мрежата не са приложени никакви мерки за регуляризация, намаляване на овърфитинга и обогатяване на данните.

Параметри на теста:

Learning rate: 0.0001

Optimizer: Adam

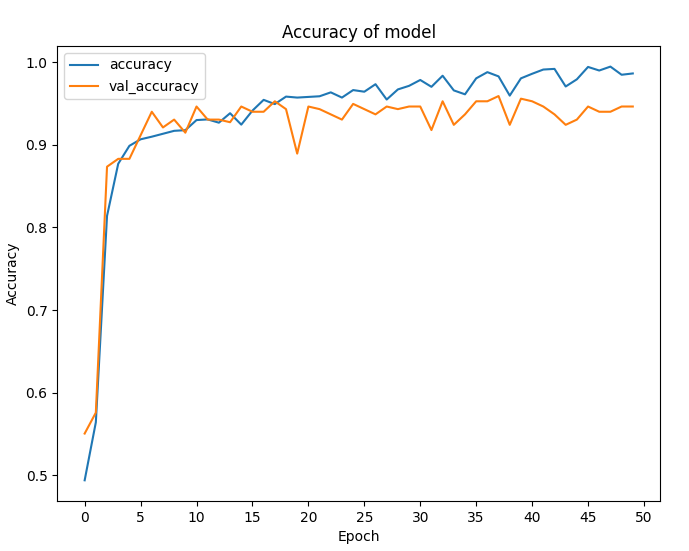
Image size: 128x128x1

Batch size: 32

Epochs: 50

Резултати:





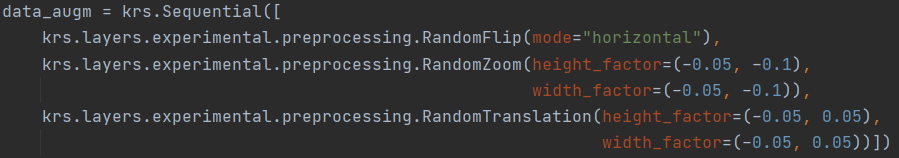
От графиките се вижда, че в процеса на обучение трениращата точност нараства, а стойността на ценовата функция намалява- мрежата се обучава успешно. Началната точност е около 50%, което в контекста на машинното обучение и изкуствения интелект се определя като “random guessing”- т.е. мрежата случайно „налучква“ прилежащия клас. Достигната точност при трениране е висока – около 98%. Добра тенденция се наблюдава и при валидационните резултати(оранжевата крива на графиките), но тя продължава до 18 eпоха. След това ясно се вижда влошаване/стагнация на резултатите при валидация, паралелно с подобряване на трениращите такива- моделът започва да се пренагажда и да се справя зле с невиждани досега данни. Достигнатата максимална валидационна точност е 95,25 % в епоха 18. В идеалният случай, кривите на валидация и трениране трябва да бъдат една върху друга. Това, разбира се, е трудно възможно, затова като допустима разлика за целите на експеримента може да се определи 0.5-1%.

Продължаващото повишаване на точността при трениране навежда на мисълта, че ако бъде намален овърфитинга, валидационната точност също ще се увеличи. Затова към архитектурата от този тест ще бъдат добавени регуляризационни техники, като dropout и обогатяване на данните.

**Тест 2.**

В този тест ще бъдат приложени техники за намаляване на пренагаждането на мрежата спрямо данните за трениране с цел да се постигне по-добра генерализация и по-висока точност при невиждани досега изображения.

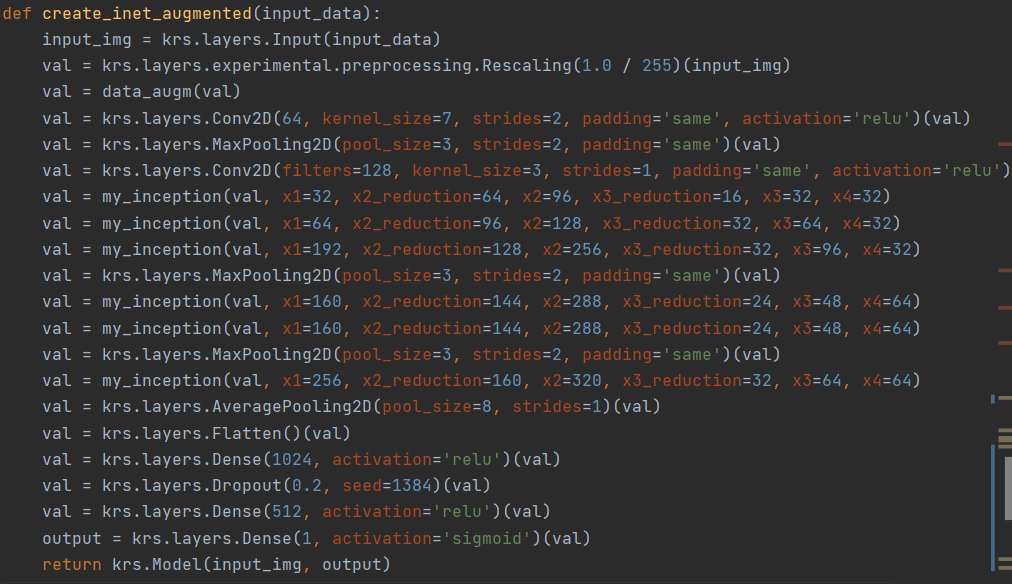
Първата такава техника е обогатяването на данните- произволни ротации, транслации и зуум на входните изображения. В текущия случай това са:

****

Това обогатяване повишава устойчивостта на модела, повишава разнообразието на входните данни и по този начин значително намалява пренагаждането.

Втората такава техника е Dropout слоят, който занулява част от входната информация, като по този начин невронната мрежа се научава да не разчита прекалено много на някои открити черти в изображенията.

След прилагане на описаните методики, архитектурата придобива следния вид:

****

Броят поддържани от мрежата параметри:



Следва трениране на новата архитектура със следните параметри:

Learning rate: 0.0001 папка inception\_v2

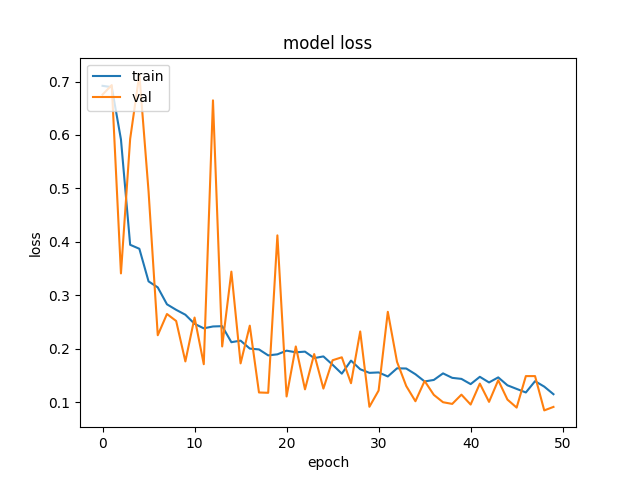
Optimizer: Adam

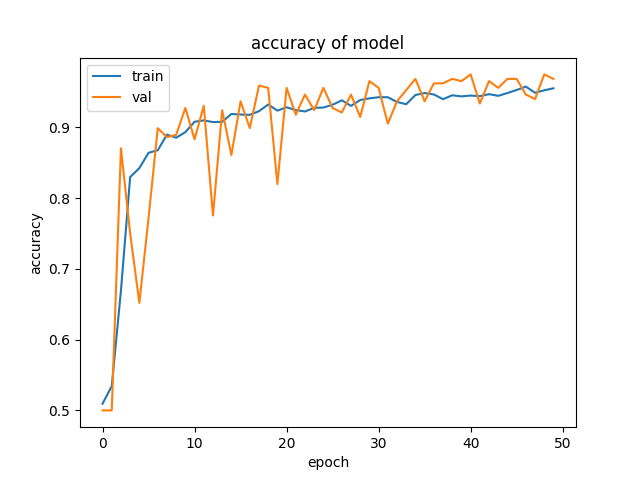
Image size: 128x128x1

Batch size: 32

Epochs: 50

Резултати от тренирането:



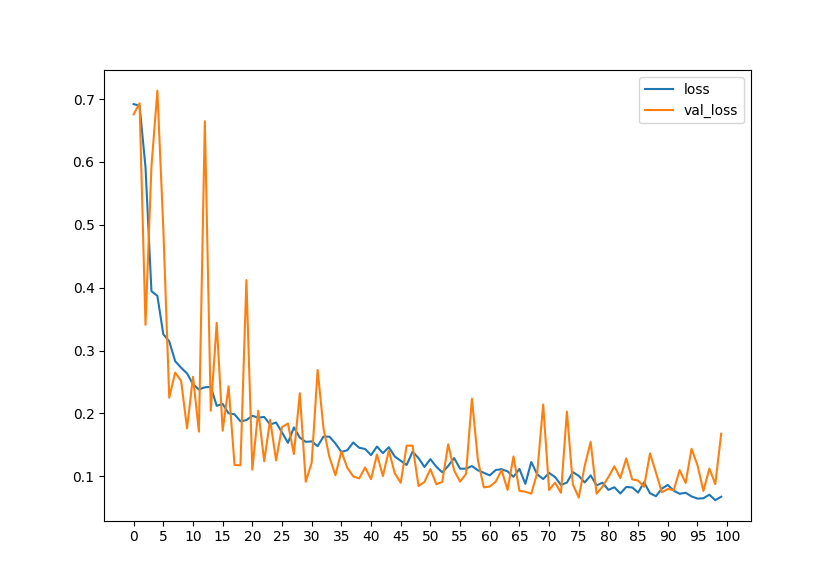


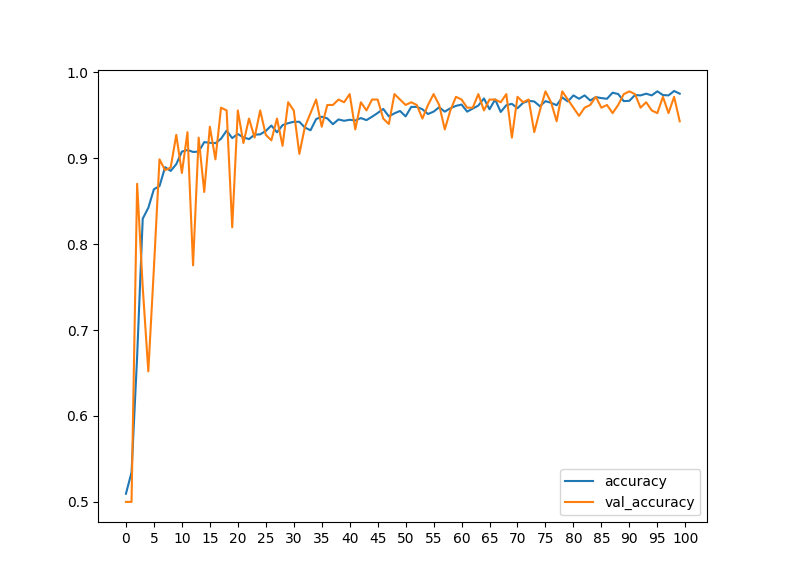
Достигнатата максимална валидационна точност е 97,47%, а трениращата 95,74%, като не се наблюдава пренагаждане на модела. В графиките впечатление прави силното осцилиране на кривите на валидационния сет. То е по-изявено в началото на трениращият процес(интервала 1-25 епоха), защото тогава промените в теглата са по-големи. За това допринася и Adam оптимизатора, който използва адаптиращи се отделни тегла за отделните параметри. Осцилирането намалява след епоха 35, защото промените в теглата и Learning rate параметрите стават все по- плавни- доближава се оптималното решение. Напред във времето на трениране се очаква още по-осезаемо стабилизиране на валидационните стойности.

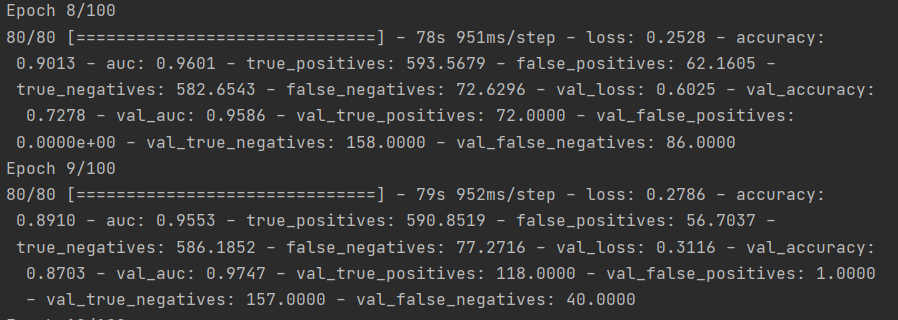
Тук се наблюдават на места по- добри резултати при валидация, от колкото при трениране, за разлика от предния тест, където валидационните резултати бяха традиционно по-лоши. Това се дължи главно на 2 неща:

* Изкуствено затруднената мрежа при трениране- **data augmentation** слоя и **dropout** слоя са **неактивни** по време на валидация. Тогава мрежата използва пълният си капацитет, при това на необогатени данни, които се очаква да класифицира по-лесно, защото не са изкуствено зашумени. Тези разлики във валидиращият/трениращият процес водят до по-добри резултати при валидация.
* По - малкият брой изображения за валидация- тренирането се извършва с 2536 изображения, докато валидацията се извършва с 316.

За да бъде направена качествена оценка на архитектурата е необходимо тренирането да продължи. Следва обучение за още 50 епохи:







Средното време за трениране на епоха е около 78 секунди. Така цялото обучение отнема около 2ч. и 10 мин.

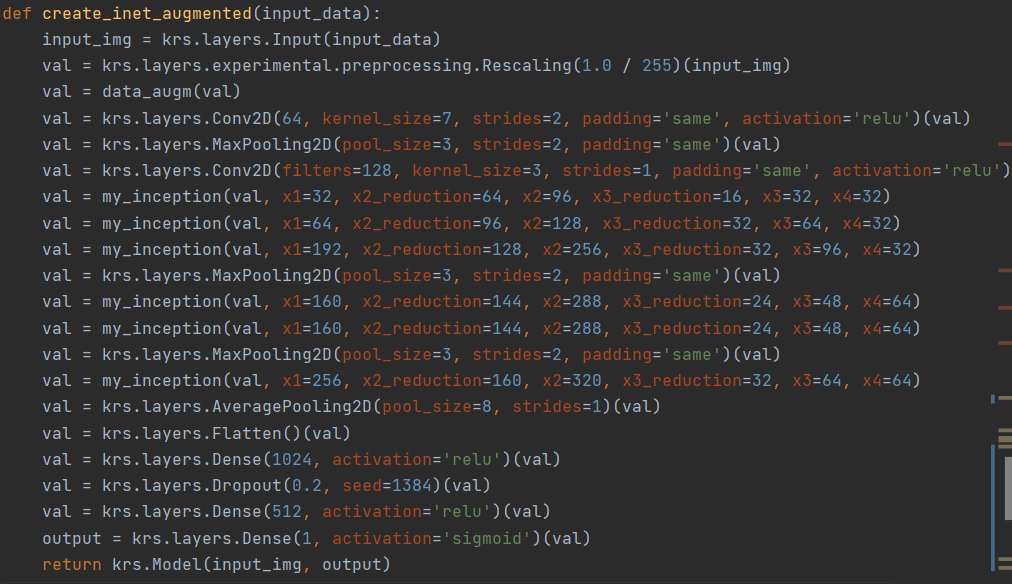
Най-високата достигната валидационна точност е 97,77% в епоха 79. В резултатите се забелязва спад на валидационната точност/увеличаване на стойността на ценовата функция след епоха 80- оградената със зелена елипса част. Може да се заключи, че мрежата е започнала да губи умението си да генерализира добре. В този случай не се налага използване на друг вид регуляризация, като L1/L2, защото достигнатата точност е достатъчно задоволителна. Процесът на трениране на текущо разглежданата архитектура се прекратява, тъй като той не би довел до по- добри резултати.

**Тест 3. Влияние на промяна на архитектурата.**

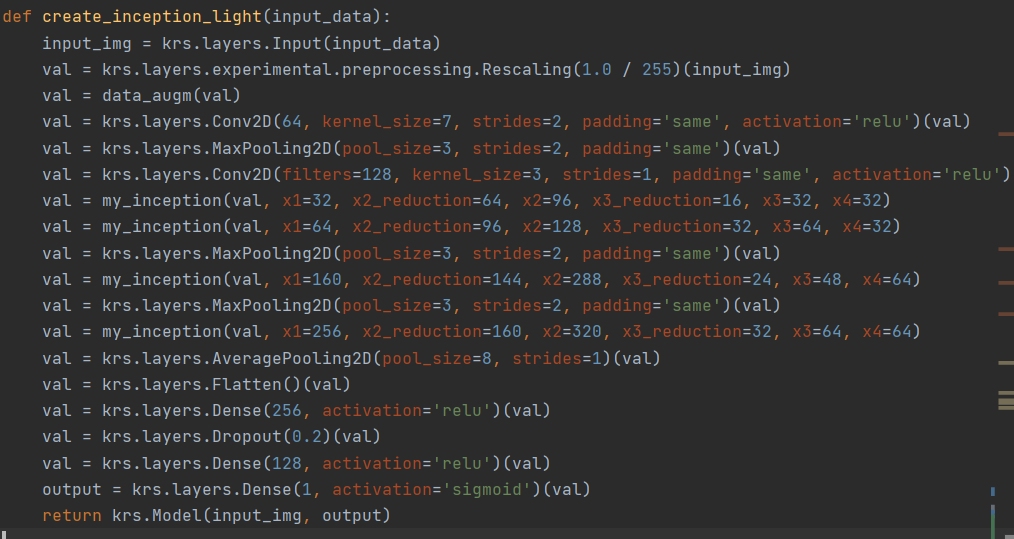
В този тест ще бъдат разгледани различни вариации на архитектурата от предния тест и как това се отразява на резултатите. Основната идея е да се провери дали може да се достигнат достатъчно добри резултати(точност върху валидиращият сет над 95%) и с по- проста архитектура. Опростяването на архитектурата трябва да бъде умерено, защото в противен случай ще се получи underfitting- невъзможност на невронната мрежа да научи спецификите на данните. Ако се приеме, че архитектурата в тест 2 е със средно ниво на сложност, тогава тя ще бъде използвана като базова. Следва да бъдат изградени по-сложен и по-опростен вариант и да се повтори процеса на трениране със всеки един от тях.

**Опростяване на невронната мрежа**

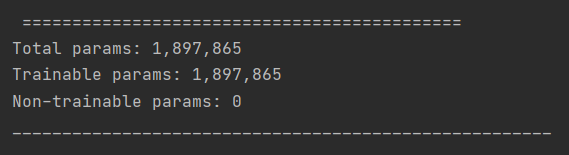
За олекотяване на мрежата ще бъдат премахнати 3 от inception модулите и ще бъде намален броят на невроните в изходните Dense слоеве съответно на 256 и 128. Премахнатите inception модули са отбелязани с червен минус на фигурата по-долу:

****

Така архитектурата приема следния вид:

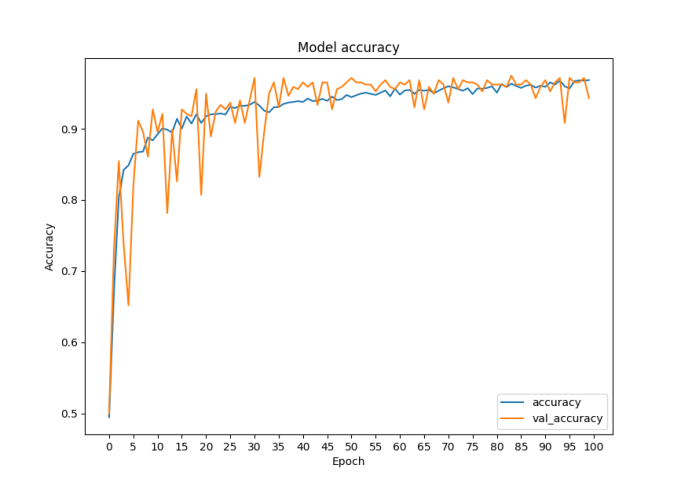
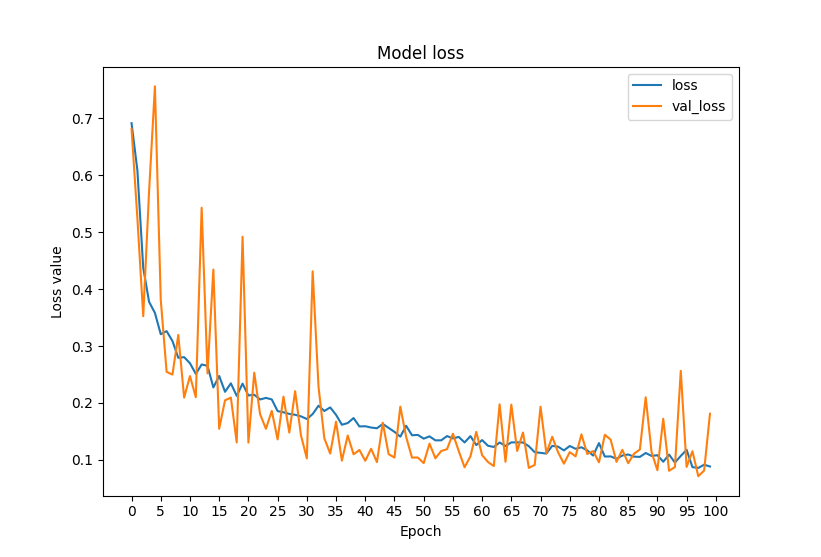


Изход от model.summary() функцията конкретния модел в Пайтън конзолата:



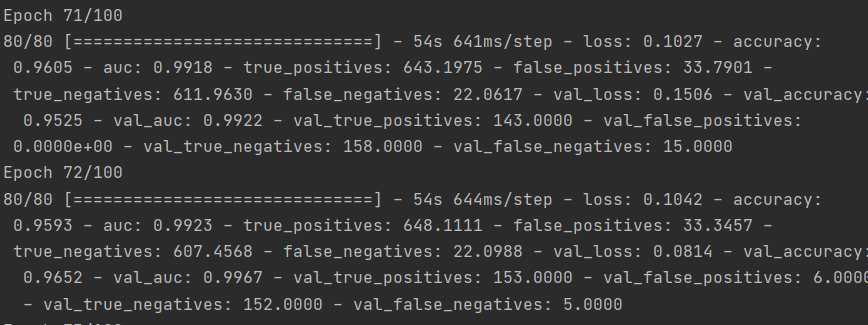
Извършено е редуциране в броя на поддържаните параметри, като броят им е сведен до около 1,9 млн. Това е около 4 пъти по-малък брой спрямо отправната базова архитектура. Следващият трениращ процес е извършен за 100 епохи със същите параметри.

Резултати:



Максималната достигната точност при валидация е 97,47% в епоха 84. След епоха 70, въпреки продължаващото повишаване на трениращата точност, във валидационната се наблюдава насищане. От това може да се заключи, че по-нататъшно обучение не би довело до по- добри резултати, а дори до спад във валидационната точност.

Средното време за трениране на една епоха с текущият хардуер, оптимизатор и learning rate е 54с. Така общото време за трениране на тази архитектура е около 1ч. и 20 мин.

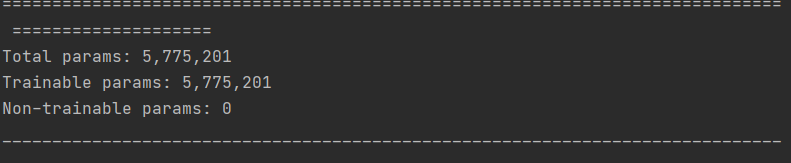


**Усложняване на невронната мрежа**

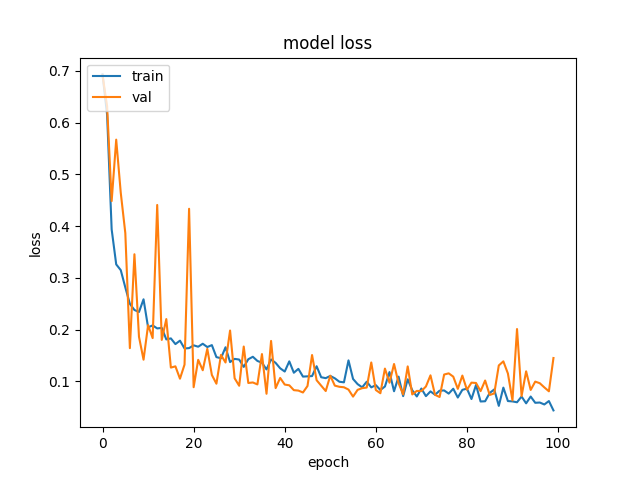
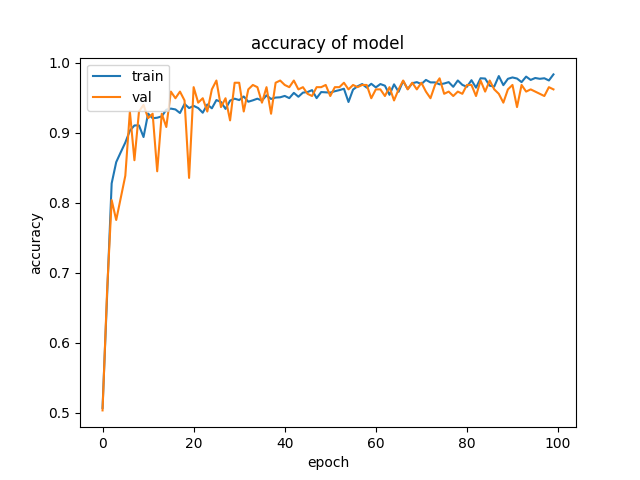
За да се постигне по-сложна архитектура са добавени още 3 на брой inception модула с подадени различен брой филтри. Броят на невроните в изходните Dense слоеве също е увеличен на 1024. Увеличаването на броя слоеве и параметри на мрежата трябва да бъде умерено, защото прекалено сложна архитектура може да доведе до ранно пренагаждане на модела и ниска валидационна точност. След уточнените промени, архитектурата придобива следния вид:



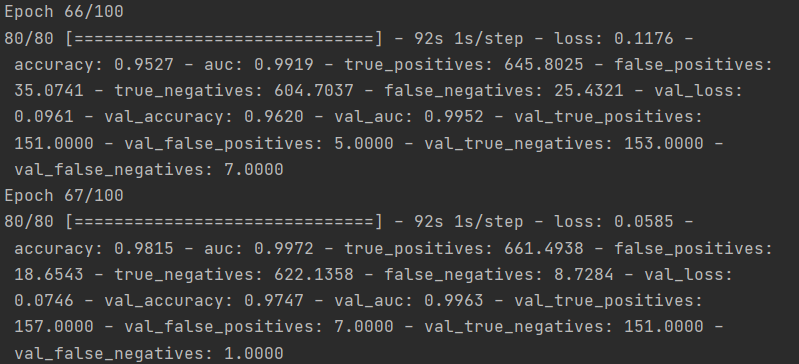
След промените, броят на поддържаните от мрежата параметри достига около 5,7 млн. Резултат от извикване на model.summary() функцията:



Отново е извършено трениране за 100 епохи със същите параметри. Резултатите от тренирането са представени по-долу:



Както се вижда от графиките, мрежата се учи успешно, но настъпва очакваното пренагаждане на модела след епоха 80- забелязва се тенденциозен спад на валидационната точност под трениращата. Максималната достигната валидационна точност е 97,78%. Въпреки леко повишената достигната точност, трябва да се отбележи, че времето за обучаване в една епоха е доста по-голямо- средно по 92 секунди:



Следователно цялото обучение за 100 епохи с текущо използвания хардуер, оптимизатор и learning rate отнема около 2 ч. и 30 мин.

**Обобщение и сравняване на резултатите**

В таблицата по-долу са обобщени резултатите от 3-те вида сложност на архитектурата на невронната мрежа:

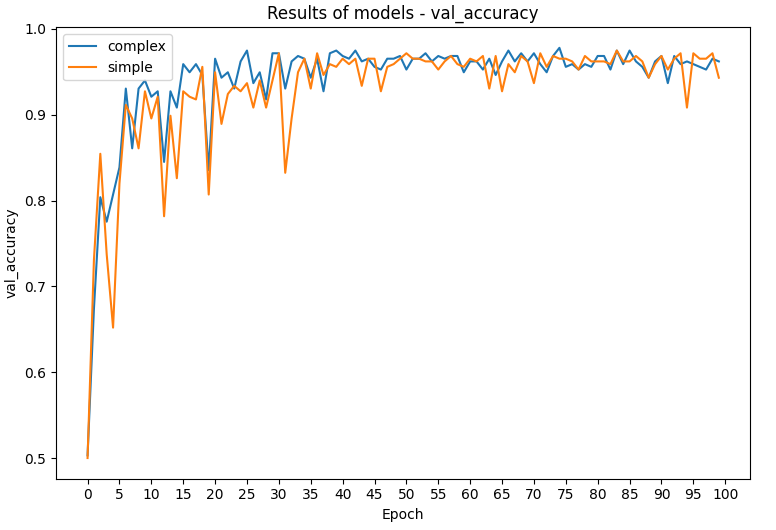
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вид архитектура | Максимална вал. точност(%) | Времетраене на обучението за 1 епоха (секунди) | Брой поддържани параметри | Брой inception модули |
| Базова | 97,77 | 78 | 4 149 825 | 6 |
| Олекотена | 97,47 | 54 | 1 897 865 | 4 |
| Усложнена | 97,78 | 92 | 5 775 201 | 8 |

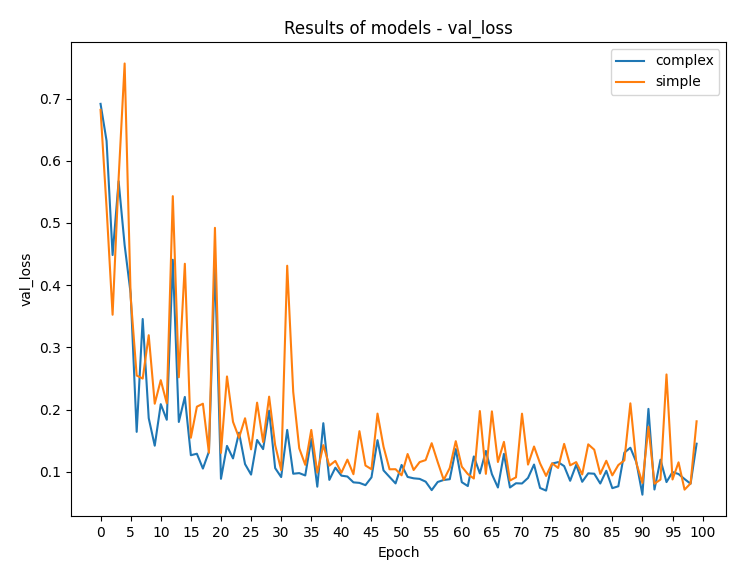
Вижда се, че и трите архитектури достигат до задоволителни стойности на валидационната точност- приблизително 97%. Очаквано, най-висока точност е достигната при най-сложната тествана архитектура- 97,78%. Разликата, обаче, спрямо базовата архитектура например, е само 0,01 %. Това е пренебрежимо малко при положение, че времето за трениране нараства с цели 14 секунди за епоха. Важен аспект при deep learning моделите е да се намери баланс между времето за трениране на модела, достигнатата точност и разход на изчислителни ресурси.

Както бе уточнено, прекалено сложните архитектури могат да доведат и до овърфитинг, като мрежата губи своята възможност да се справя с невиждани до сега данни- пренагаждане се случи и при трениране на усложнената архитектура, при което във валидационната точност се наблюдава спад, за разлика от трениращата точност. Освен това, след имплементацията на усложнената архитектура, тя трябва да поддържа 5,7 млн. параметри- около 1,6 млн. повече спрямо базовата такава. Това е значително по-висок разход на ресурс на компютърната система.

Олекотената архитектура също постига много добри резултати- 97,47% точност на валидиращия сет. Големи нейни предимства са ниското време за обучение на епоха и сравнително ниския брой поддържани параметри- 1.8 млн., което намалява значително и изискванията към памет/изчислителна мощ. Но използваният дейтасет за обучение не съдържа огромен брой изображения(като например Imagenet сета), а около 2600 изображения за трениране. Възможно е при използване на много по-голям набор от изображения олекотената архитектура **да не може** да усвои напълно новопоявили се важни характеристики в изображенията и да се справя доста по-лошо.

По-долу са плотирани в една и съща к.с. валидационната точност/стойността на ценовата функция за усложнения(complex) и опростения(simple) вариант на архитектурата. На тях ясно се вижда, че по- богатата архитектура достига по-висока точност и по- ниска loss стойност. Усложненият вариант дори в началните епохи достига до точност над 95%. Впечатление прави и намаленото осцилиране в графиките при по-комплексната мрежа спрямо това при опростената.





С оглед резултатите от тренирането и тестването на трите архитектури, смятам че базовата е най-подходяща за използване в случая, защото постига точност, много близка до тази на сложната архитектура, но използва по-малко параметри и времето ѝ за трениране е по-кратко. В сравнение с олекотената архитектура, смятам, че базовата има предимството по-лесно и достатъчно точно да улови новооткрити черти в изображенията, когато в по-късен етап дейтасета бъде разширен с повече изображения. Затова в следващите тестове ще бъде използвана именно базовата.

**Тест 4. Влияние на промяна на learning rate.**

В този тест ще бъде проследено влиянието на различните стойности на learning rate хиперпараметъра. Подбирането на learning rate(LR) е важна част от трениращият процес, оказващ силно влияние върху намирането на оптималното(близко до оптималното) решение. При използване на прекалено голяма стойност за LR, оптимизатора най-често прескача оптималното решение – т.нар. optimum overshooting. В такива случаи се наблюдава силно осцилиране в графиките на точността/ценовата функция- било то по време на валидация или трениране. От друга страна, при използване на прекалено малка стойност на LR се случват 2 нежелани явления:

* Бавно трениране- обучителния процес се извършва на по-малки стъпки, следователно е и по-бавен
* Засядане в локален минимум- прекалено малката стойност на LR може да доведе до стагнация на обучителния процес, поради невъзможност за излизане от локален минимум.

Именно затова е важно да бъде подбрана подходяща стойност за learning rate параметъра, различаващ се за различните проблеми.

В предните тестове е използван Adam оптимизатора- един от т.нар. адаптивни оптимизатори, при който learning rate се адаптира за всеки параметър спрямо честотата на неговата промяна. Така в контекста на Adam оптимизатора, подаденият learning rate параметър се явява горна граница при адаптацията. Ще бъдат тествани няколко стойности за LR, a като архитектура на невронната мрежа ще бъде използвана „базовата“ от предният тест. Learning Rate параметърът от предишните тестове- 0.0001 показва ефективно обучаване на невронната мрежа. Следователно е добре тази стойност да се приеме за базова отправна точка и да се провери резултатът от намаляването/увеличаването ѝ.

Run 1:

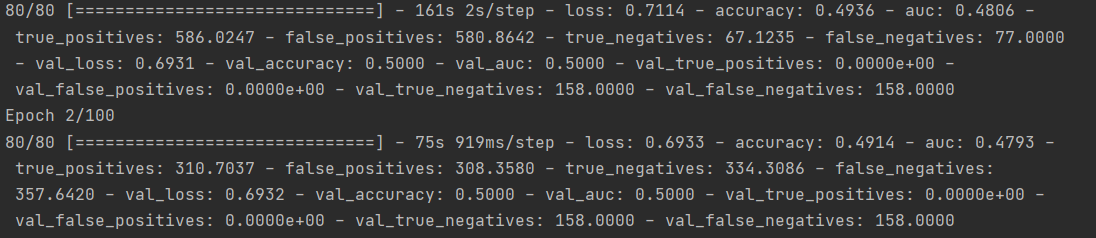
Трениране с по-голяма стойност на learning rate от базовата.

Параметри на обучителния процес:

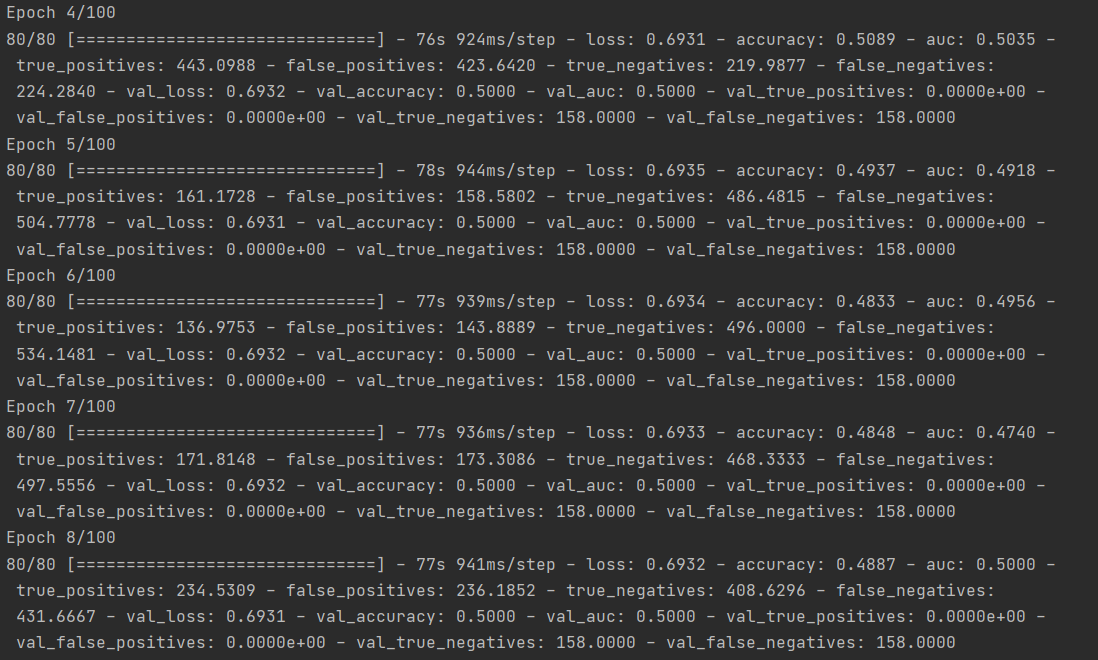
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Optimizer** | **Learning rate** | **Image Size** | **Batch Size** | **Epochs** |
| Adam | 0.001 | 128x128x1 | 32 | 100 |

Резултати:

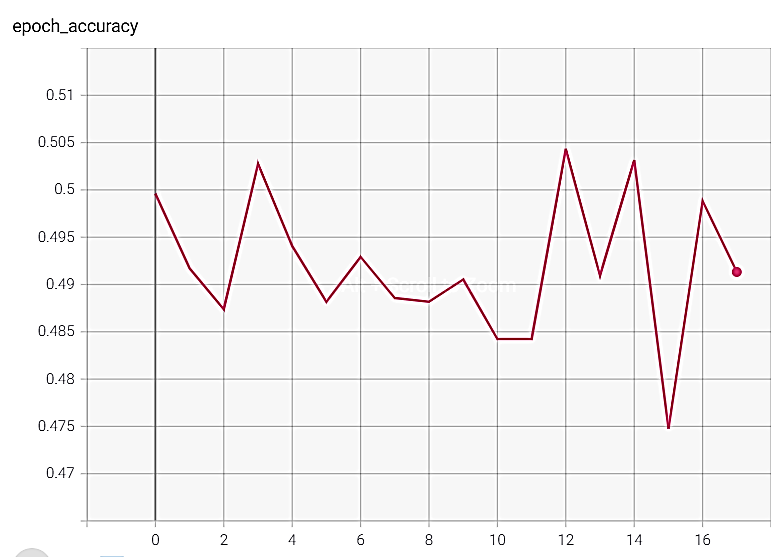
По време на обучителния процес се забелязва, че в началните епохи, точността е много ниска- около 50%:



В този случай мрежата извършва т.нар. Random Guessing, като процеса на класификация на изображенията може да се сравни с хвърляне на монета „ези-тура“. Това явление се наблюдаваше и в предните обучителни процеси(с други стойности на LR)- **през първите 2-3 епохи,** след което точността **се повишава.** При тренирането със стойност 0.001, обаче, се случва друго:



От фигурата, на която е показан изхода от Пайтън конзолата по време на трениране се вижда, че дори с напредване на обучителния процес, точността варира между 51 и 48% с тенденция за спад напред във времето. Същевременно се наблюдава стагнация в стойността на loss функцията. Това се вижда и на графиките за трениращата точност:



Training accuracy

Epoch

****

Epoch

Training loss

В този случай следва да се прекрати преждевременно тренирането, защото мрежата не се обучава ефективно. LR се оказва прекалено висок и въобще не се върви към посока оптимално решение. Същите тенденции се наблюдават и при трениране с намалена на 0.0005 стойност за learning rate параметъра. Така може да се заключи, че максималната стойност, която води до ефективно обучение на мрежата с точно тези параметри и оптимизатор, е използваната и в предните тестове – 0.0001. При опити за увеличаване на LR, тренирането на мрежата е без успех, както се вижда от фигурите по-горе.

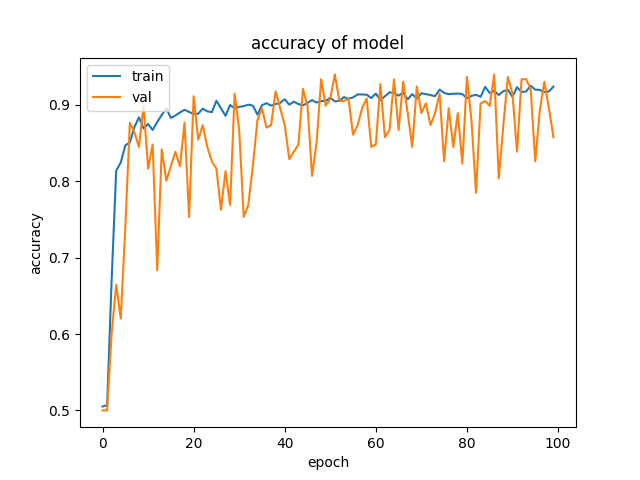
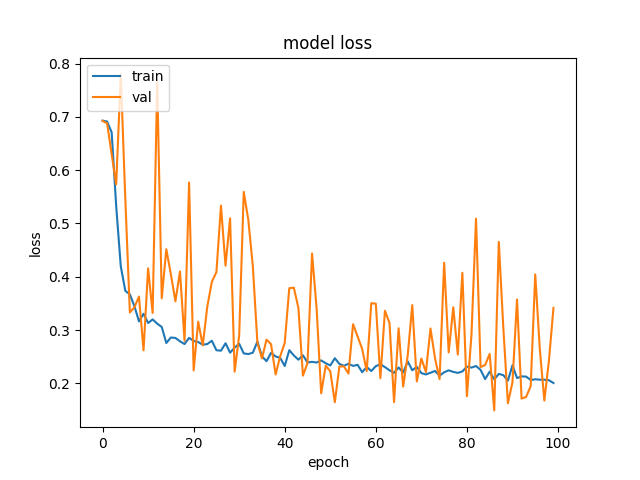
**Run 2:**

Извършено е намаляване на learning rate спрямо базовия на стойност 0.00001. Това би следвало да доведе до по-плавни промени при обновяване на теглата, но за сметка на по-бавно достигане на близко до оптималното решение.

Параметри на теста

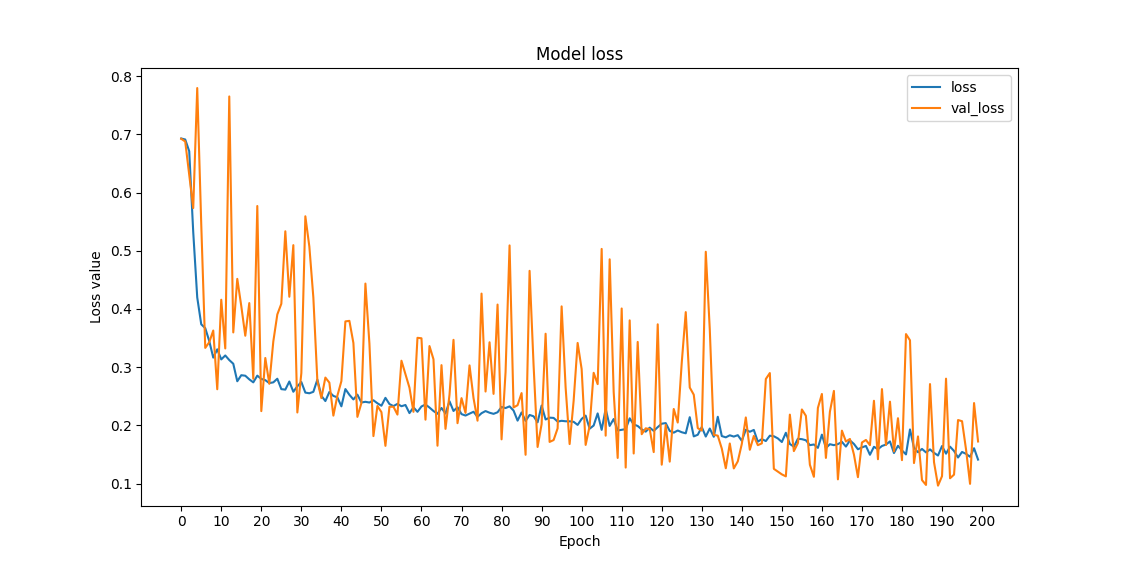
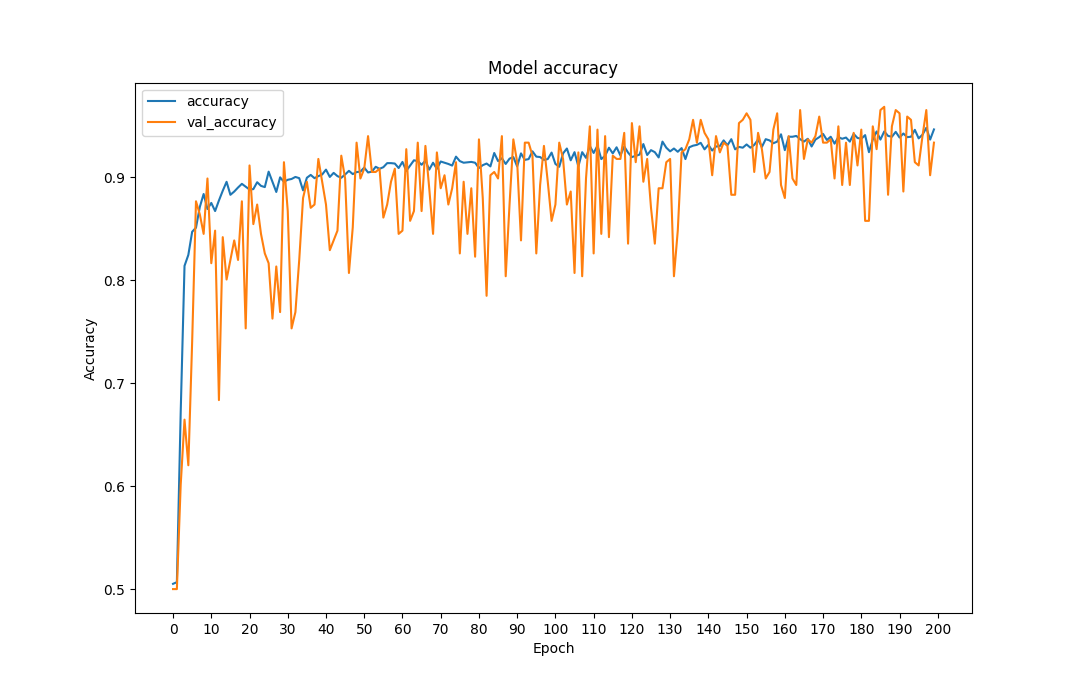
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Optimizer** | **Learning rate** | **Image Size** | **Batch Size** | **Epochs** |
| Adam | 0.00001 | 128x128x1 | 32 | 100 |

**Резултати:**

****

От графиките за точността и стойността на целевата функция се вижда, че трениращата точност расте, макар и на по-малки стъпки, което води до извода, че мрежата се обучава успешно. Достигнатата максимална точност при валидация е 93% в епоха 47. Осцилирането във валидационните криви е доста по-изявено спрямо наблюдаваното при базовата стойност на LR(0.0001). Това е така, защото е нужно повече време за стабилизиране на LR на отделните параметри заради динамичния характер на използвания оптимизационен метод Adam. Между епоха 40 и 60 се наблюдава лека стабилизация на валидационната точност, последвана от леко влошаване на резултатите. Нужно е тренирането да продължи за още 100 епохи, за да се провери тенденцията.

Резултати:



Въпреки временното влошаване на точността, мрежата продължава да се обучава ефективно. След епоха 130 се наблюдава значително стабилизиране в стойностите. Достигната точност при валидация е 96,84% в епоха 187. Loss стойността при трениране продължава да спада, което значи, че мрежата може да продължи обучението си, с риск, разбира се, да настъпи пренагаждане на модела. Понижената стойност на LR, обаче, води до доста по-бавно достигане до задоволителни резултати в сравнение с базовата стойност.



Това се вижда и на фигурата, изобразяваща кривите на точността при LR= 0.0001 и LR= 0.00001 в една к.с. Така например, в епоха 15 разликата е около 5%. Личи и колко по-стръмна е кривата на базовия learning rate(на фигурата в син цвят)- обновяванията на теглата са по- резки и по бързо се достига оптимално решение. В таблицата по-долу са представени резултатите от обучението с различните стойности на LR:

|  |  |  |  |
| --- | --- | --- | --- |
| Стойност на хиперпараметър  Learning Rate | Достигната валидационна точност(%) | Брой епохи | Успешно обучаване |
| 0.001 | 49,89 | 30 | Не |
| 0.0001(база) | 97,77 | 100 | Да |
| 0.00001 | 96,84 | 200 | Да |

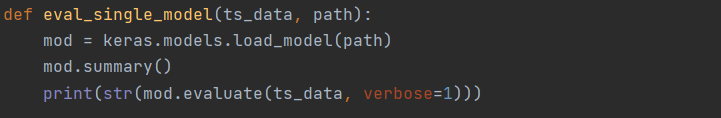
С оглед проведените тестове с различни стойности на LR хиперпараметъра, най- подходяща се оказва базовата 0.0001, защото намира баланса между по-бързото обучение и същевременно достигане до близко до оптималното решение. След достатъчно продължително трениране на мрежата с LR=0.00001, в крайна сметка ще се достигне близко до оптималното решение. Възможно е точността и нивото на грешките да е по- добро, в сравнение достигнатата при базовия LR. В случая, това се явява непрактично, защото времето за трениране на мрежата се покачва неколкократно- дори и при 2 пъти по- продължително обучение, достигнатата максимална точност е по- ниска от базовата.

**Финална оценка**

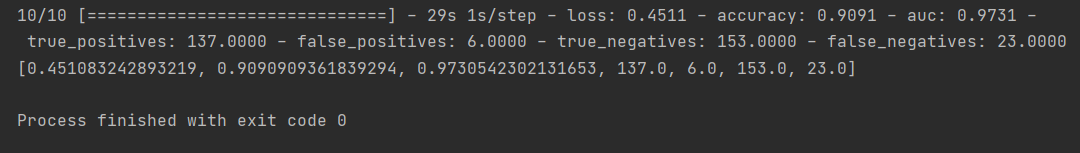
Финалната оценка се състои в тестване на най-добрите модели от Опит 1 и Опит 2 върху тестовия сет от данни. Той е с размер 10% от всички данни, или 319 изображения.

**Оценка от Опит 1**

Оценката ще бъде извършена с регуляризираният модел(L2=0.00001), епоха 59. В това състояние мрежата достига валидационна точност от 95,25%. Следва да се провери тестовата точност върху тестовия сет. Пайтън скрипт:



Резултат конзолата:



Достигнатата точност върху тестовият сет е 90.91%. Въпреки, че е доста висока, сама по себе си, тя не е достатъчна за адекватна оценка. Под внимание трябва да се вземат и :

* Действително положителните класификации true positives(TP) - 137
* Действително отрицателните класификации true negatives (TN) – 153
* Фалшиво отрицателни класификации false negatives (FN) – 23
* Фалшиво положителни класификации false positives (FP) – 6

На база тях може да се изчислят чувствителността - True positive rate (TPR), специфичността True negative Rate(TNR) и прецизността Positive predictive value(PPV):

**TPR= TP/TP+FN= 0.856**

**TNR= TN/TN+FP= 0.962**

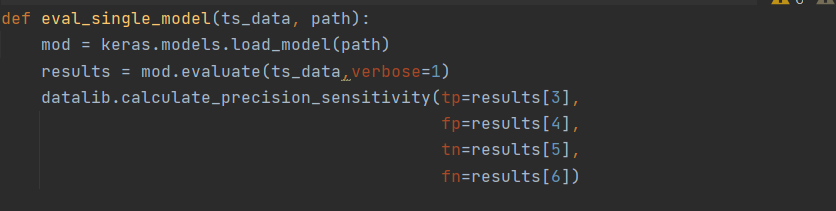
**PPV= TP/TP+FP= 0.958**

Получената стойност за чувствителността е задоволителна – 0.856 (85,6 %). В случая тя изразява колко добре конструираният модел може да класифицира правилно хора към клас 1(Pneumonia), действително имащи заболяването т.е. True positives. Получената стойност за специфичността е сравнително висока – 0.962 (96,2 %). Тя репрезентира колко добре моделът класифицира здравите хора към клас 0(Normal), т.е True Negatives. Съществува компромис(tradeoff) между чувствителността и специфичността – при висока чувствителност се наблюдава по-ниска специфичност и обратното. В случая мрежата има по- висока специфичност, което би довело до по-малко фалшиво положителни резултати. Така се намалява нуждата от допълнителни ненужни изследвания на неправилно класифициран пациент.

**Оценка от Опит 2**

Ще бъде извършена оценка с тестовият дейтасет на избран от Опит 2 модел, използващ Inception модула. За оценката ще бъде използвана базовата архитектура, защото тя даде балансиран резултат по време на обучителният процес. Избрано е състоянието от епоха 79, достигащо максимална валидационна точност от 97.77%.

Тестване с тестовия сет, като отново се изпълнява скрипта за оценка с подаден пътят към записаното състояние на модела в епоха 79:



Резултат:



Както се вижда, достигнатата точност на тестовия сет е много висока – 96.87%. Разликата между нея и валидационната е дори по-малка от 1%. Броят на фалшиво отрицателните и фалшиво положителните класификации е много малък- съответно 7 и 3. Така за прецизността, чувствителността и специфичността се получават:

**TPR= 0.956**

**TNR= 0.981**

**PPV= 0.9808**

За избраната конфигурация на мрежата, отново специфичността(TNR) е по- висока, но се наблюдава значително подобрение в чувствителността(TPR) спрямо оценения от Опит 1 модел. Предимство е, че в случая не се наблюдава явлението „overfitting на валидационните данни“- тестовата точност е много близка до валидационната.

В заключение, подбраният модел от Опит 2 се справя отлично с невиждани досега данни, постигайки удивителни резултати върху тестовия сет. Моделът успява да генерализира достатъчно добре, поддържайки висока стойност на чувствителността от 95.6%, същевременно подкрепена от висока стойност на прецизността- 98.1%, показваща високо ниво на достоверност на резултат „има наличие на пневмония“. На база изброените предимства пред тестваният от Опит 1 модел, както и показаните отлични резултати, тестваният от Опит 2 модел се избира за окончателен. Така той може ефективно и пълноценно да бъде вграден и използван в реално приложение, реализиращо компютърно-подпомогната медицинска диагностика.

Източници: TODO

<https://medium.com/strands-tech-corner/unbalanced-datasets-what-to-do-144e0552d9cd>