**Опит 1.**

Използвана архитектура на невронната мрежа- Modified Xception- хибрид между Inception и ResNet от keras.io – source code:

<https://keras.io/examples/vision/image_classification_from_scratch/>

* резолюция на изображенията- 256x256 пиксела, заредени в grayscale
* Batch size – 8 изображения
* Брой изображения за трениране на мрежата – 2668, като е зададен 20% validation split за автоматична валидация след всяка епоха. При всяко ново зареждане на изображенията за трениране в паметта и респективно заделянето на сет за валидация е използван един и същ **seed(1333)-** винаги едни и същи изображения попадат в частта за валидация/трениране
* Използван хардуер за изчисленията – GPU – NVIDIA GeForce 940mx – 2GB ; Memory Bandwidth - 37,33 GiB/s ; ComputeCapability – 5.0

Код на архитектурата на мрежата:

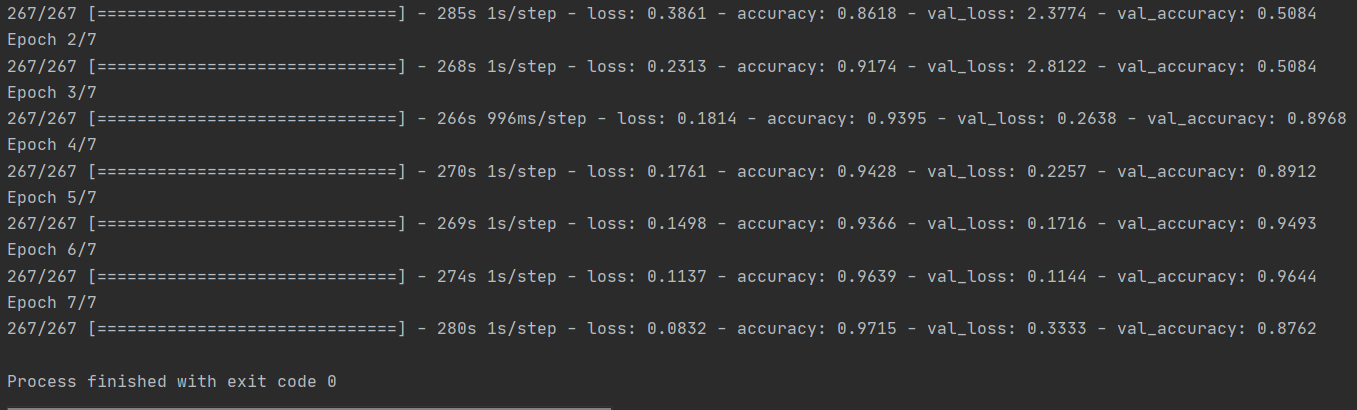
def model\_create\_xception(input\_shape, num\_classes):  
 inputs = krs.Input(shape=input\_shape)  
  
 # Entry block  
 x = krs.layers.experimental.preprocessing.Rescaling(1.0 / 255)(inputs)  
 x = krs.layers.Conv2D(32, 3, strides=2, padding="same")(x)  
 x = krs.layers.BatchNormalization()(x)  
 x = krs.layers.Activation("relu")(x)  
  
 x = krs.layers.Conv2D(64, 3, padding="same")(x)  
 x = krs.layers.BatchNormalization()(x)  
 x = krs.layers.Activation("relu")(x)  
  
 previous\_block\_activation = x # Set aside residual  
  
 for size in [128, 256, 512, 728]:  
 x = krs.layers.Activation("relu")(x)  
 x = krs.layers.SeparableConv2D(size, 3, padding="same")(x)  
 x = krs.layers.BatchNormalization()(x)  
  
 x = krs.layers.Activation("relu")(x)  
 x = krs.layers.SeparableConv2D(size, 3, padding="same")(x)  
 x = krs.layers.BatchNormalization()(x)  
  
 x = krs.layers.MaxPooling2D(3, strides=2, padding="same")(x)  
  
 # Project residual  
 residual = krs.layers.Conv2D(size, 1, strides=2, padding="same")(  
 previous\_block\_activation  
 )  
 x = krs.layers.add([x, residual]) # Add back residual  
 previous\_block\_activation = x # Set aside next residual  
  
 x = krs.layers.SeparableConv2D(1024, 3, padding="same")(x)  
 x = krs.layers.BatchNormalization()(x)  
 x = krs.layers.Activation("relu")(x)  
  
 x = krs.layers.GlobalAveragePooling2D()(x)  
 if num\_classes == 2:  
 activation = "sigmoid"  
 units = 1  
 else:  
 activation = "softmax"  
 units = num\_classes  
  
 x = krs.layers.Dropout(0.5)(x)  
 outputs = krs.layers.Dense(units, activation=activation)(x)  
 return krs.Model(inputs, outputs)

**Run № 1:**

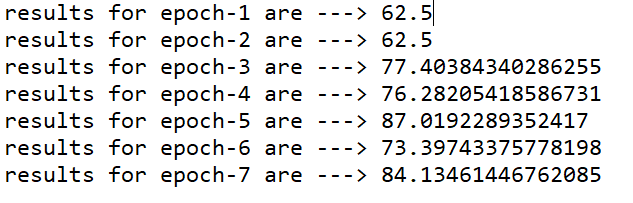
Параметри:

optimizer- Adam(learning\_rate= 0.001) ; loss function- binary crossentropy ; **number of epochs – 7**

Резултати при тренирането:



Резултати при тестването:



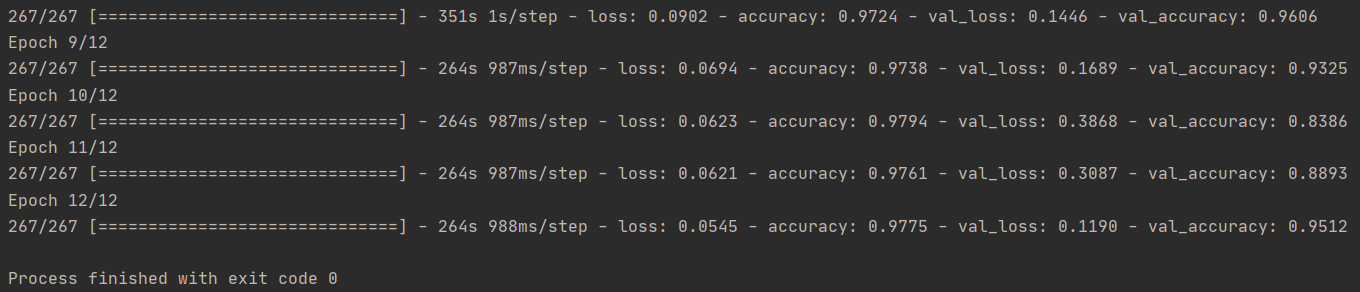
**Run № 2:**

Зареждане на модела от .h5 файл в състояние мрежата- епоха 7 и трениране за още 5 епохи

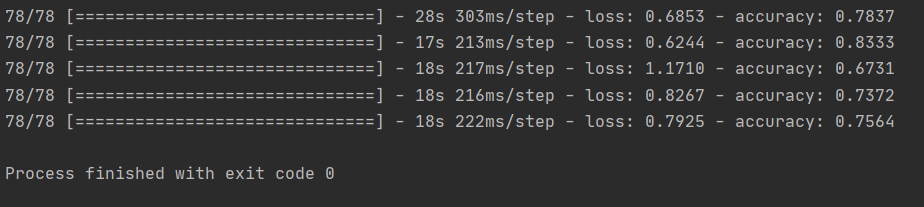
Параметри:

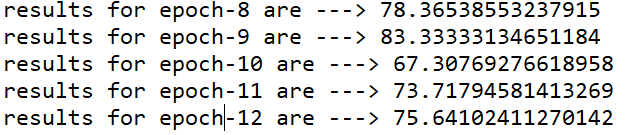
optimizer- Adam(learning\_rate=0.001) ; loss function- binary crossentropy ; **number of epochs – 5(total of 12)**

Резултати при тренирането:

****

Резултати при тестването:



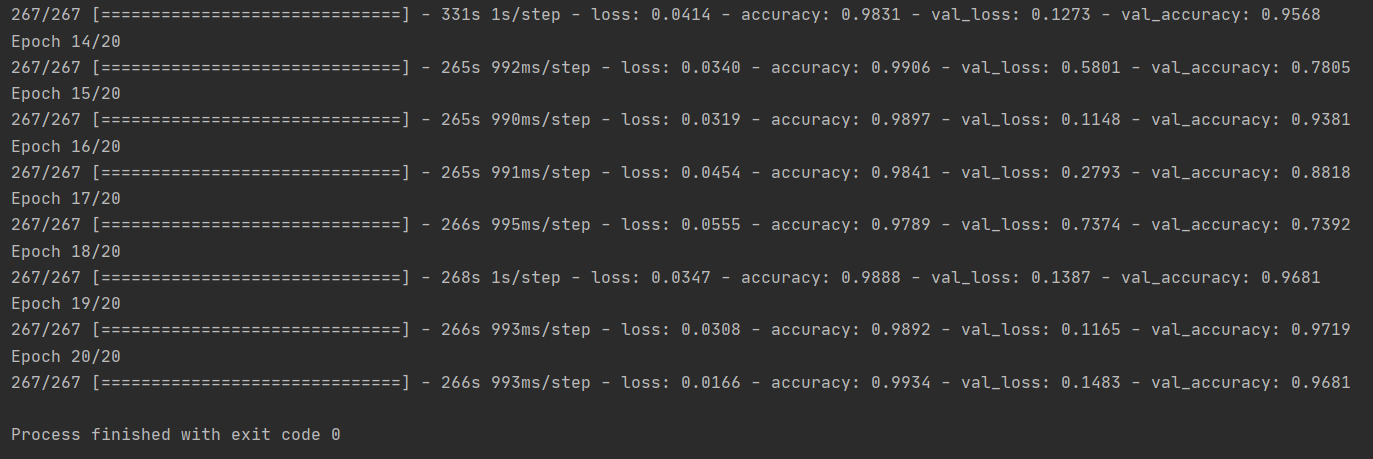


**Run № 3:**

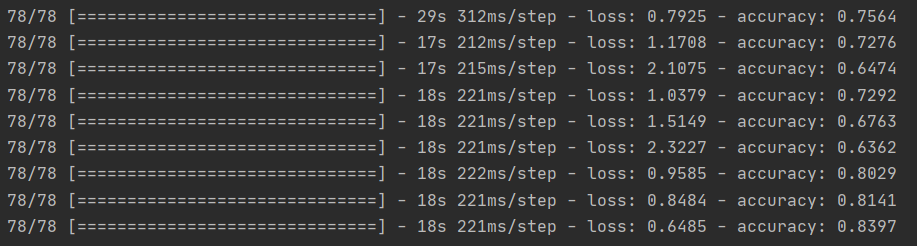
Зареждане на модела от .h5 файл в състояние мрежата- епоха 12 и трениране за още 8 епохи

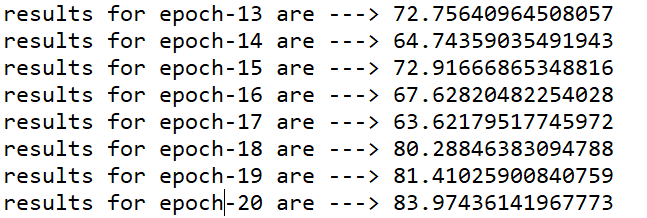
Параметри:

optimizer- Adam(learning\_rate=0.001) ; loss function- binary crossentropy ; **number of epochs – 8(total of 20)**



Резултати от тестването:



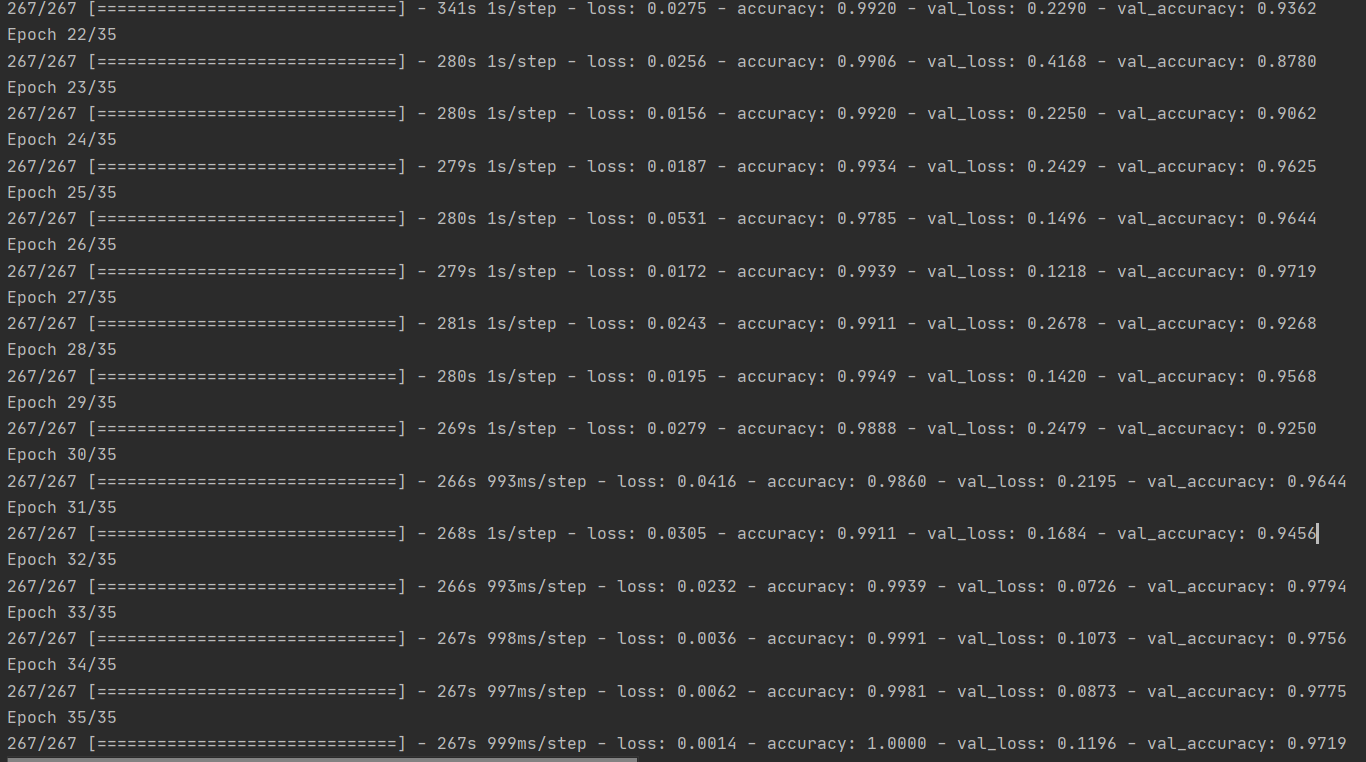


**Run № 4:**

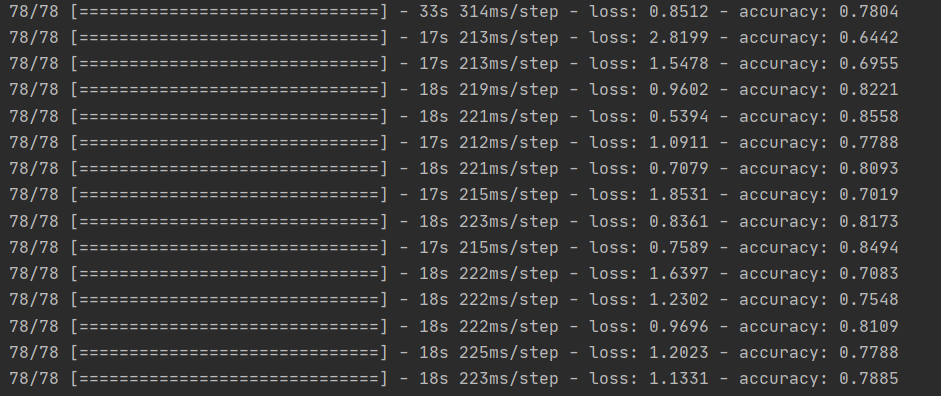
Зареждане на модела от .h5 файл в състояние мрежата- епоха 20 и трениране за още 15 епохи

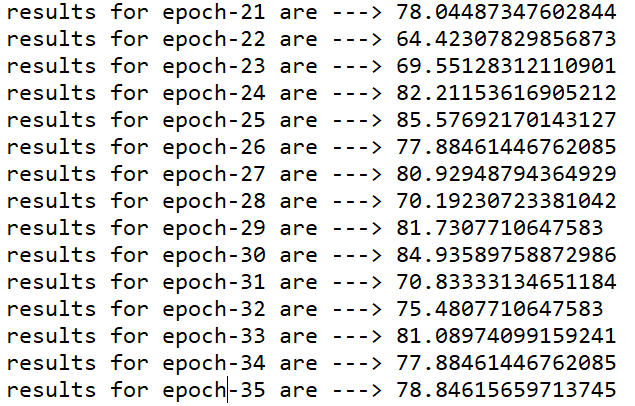
Параметри:

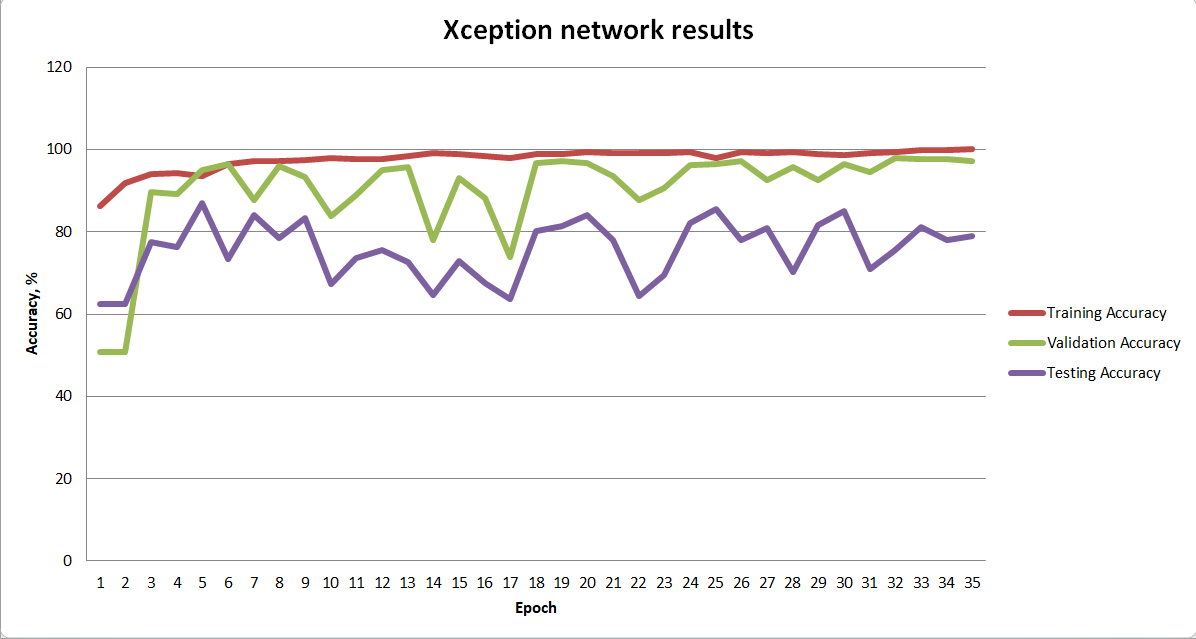
optimizer- Adam(learning\_rate=0.001) ; loss function- binary crossentropy ; **number of epochs – 15(total of 35)**

****

Резултати от тестването :







След епоха 32 се наблюдава насищане на стойността на валидационната точност- около 97% в 4 последователни епохи. Забелязва се и голяма разлика между точността при трениране и тестване на мрежата, което сигнализира за доста вероятен overfitting на модела спрямо данните за трениране.

**Run № 5:**

Зареждане на модела от .h5 файл в състояние мрежата- епоха 35 и трениране за още 10 епохи

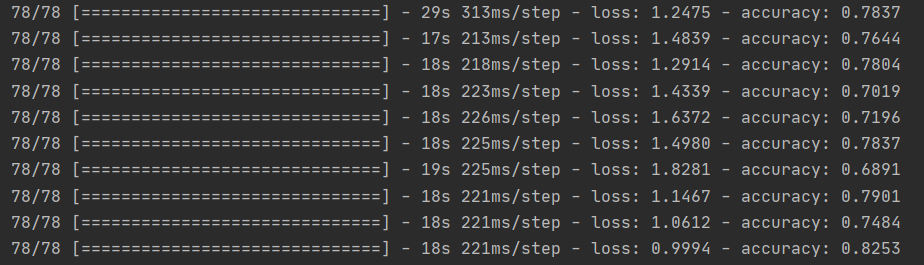
Параметри:

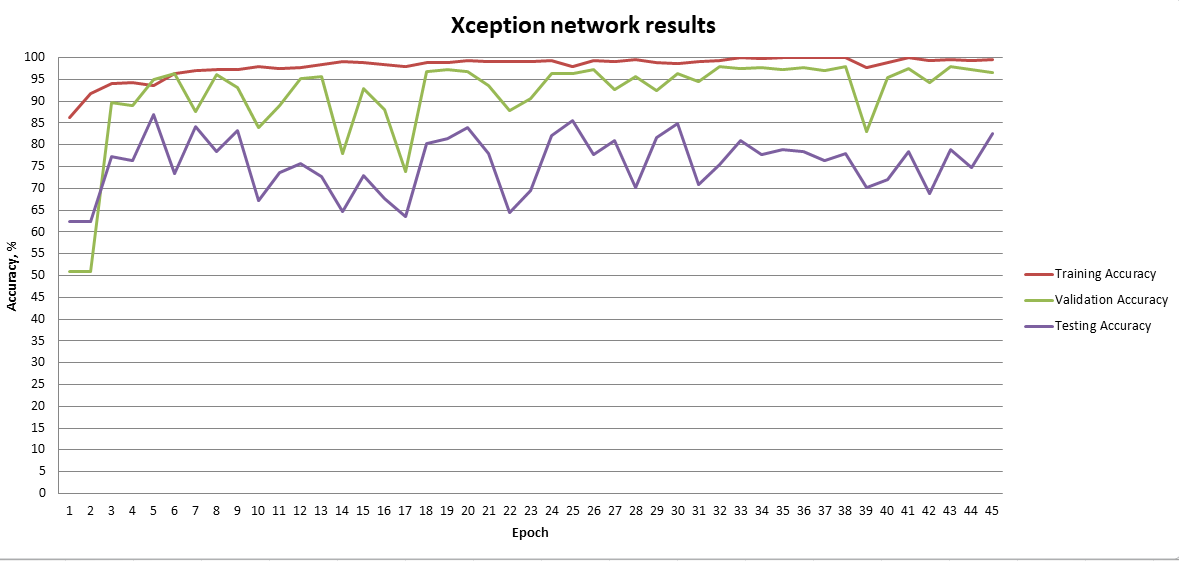
optimizer- Adam(learning\_rate=0.001) ; loss function- binary crossentropy ; **number of epochs – 10(total of 45)**

Резултати при тренирането:



Резултати при тестването:





Наблюдава се и осцилиране на валидационната точност. Възможна причина може да бъде малкият размер на едновременно разглеждани изображения(batch size) – 8. Тренирането на мрежата е извършено без data augmentation. За преодоляване на осцилирането и намаляване на overfitting-a може да бъдат предприети следните мерки:

1. Увеличаване на batch size параметъра(в рамките на разумното и до колкото позволява хардуерът)

2. Промяна на learning rate параметъра с по- малка стойност

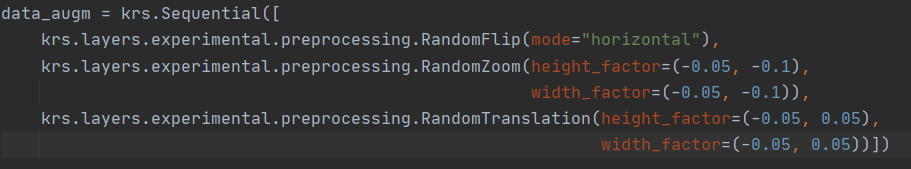
3. Промяна в архитектурата на мрежата.

4. Добавяне на „обогатяване на данните“ - data augmentation

**Опит 2:**

При опит 2 са приложени следните мерки за ограничаване на осцилирането и overfitting-a:

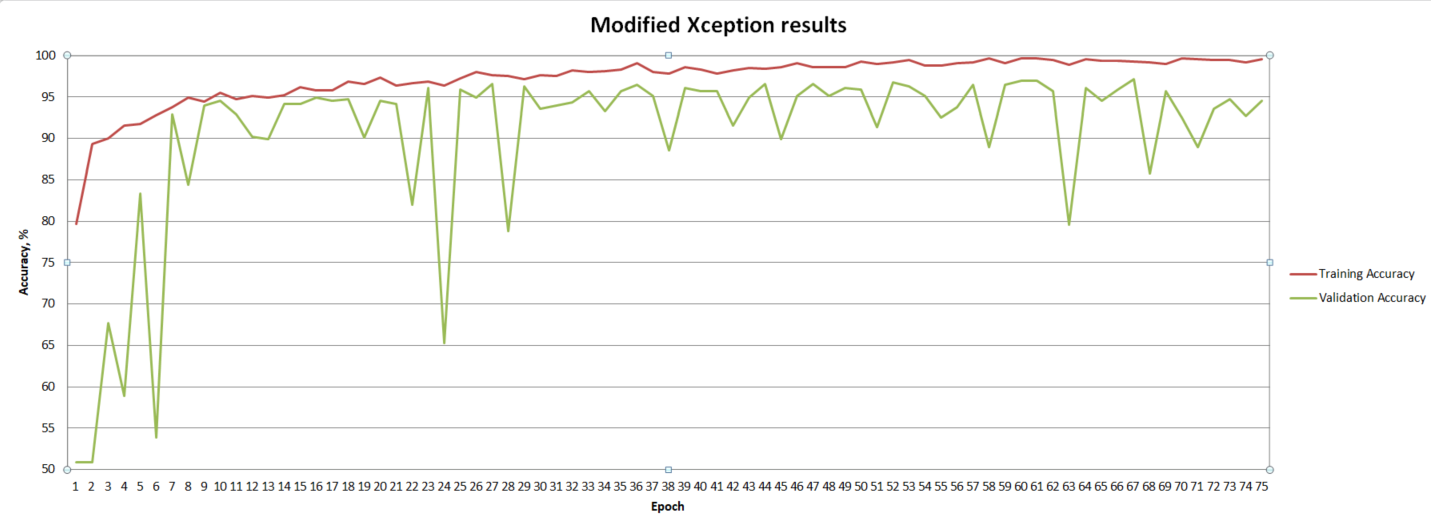
* Добавяне на data augmentation **към архитектурата на невронната мрежа(**обогатяването се извършва „on-line“ върху GPU като слоеве от мрежата**)**

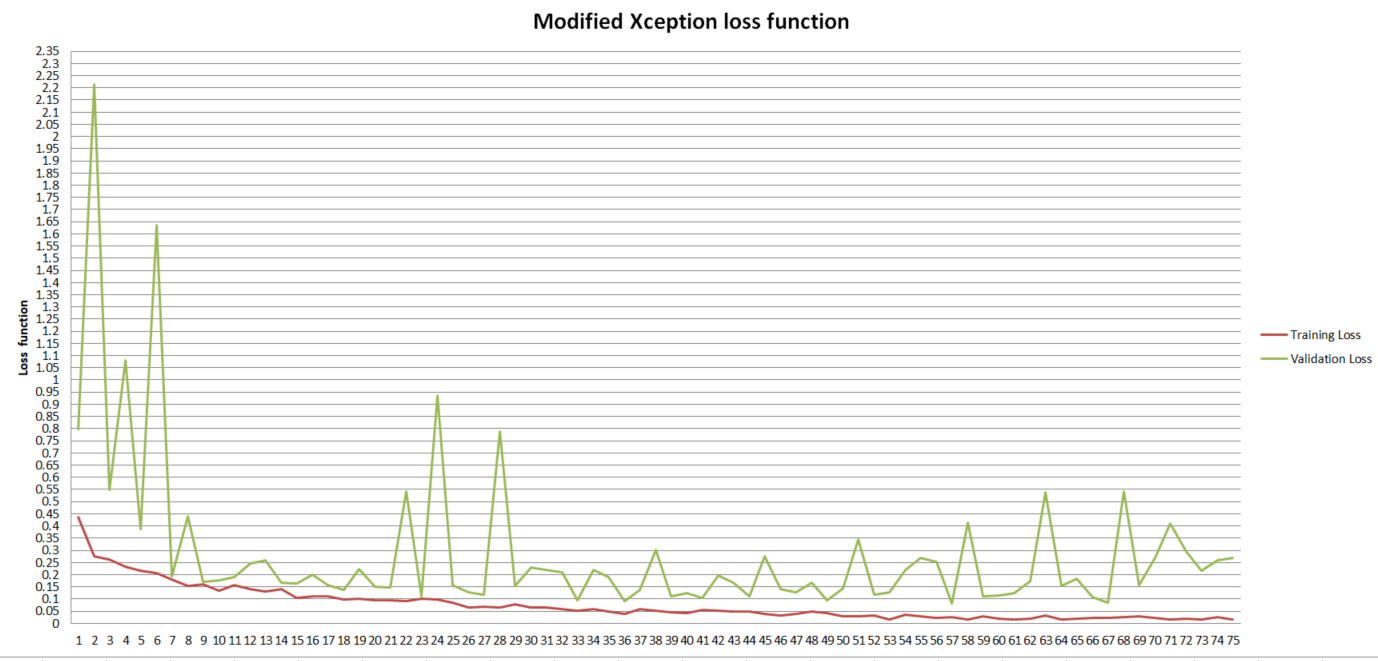


* Промяна на стойността на вероятността на DropOut слоя в мрежата на 0.6 (предишна стойност – 0.5)
* Промяна на хиперпараметъра learning rate на оптимизационния метод Adam на 0.00005(предишна стойност 0.001)
* Стойността на batch size е запазена същата- batch size= 8
* Зададено е отпечатване на следните метрики след всяка епоха – Accuracy, AUC, TP, FP, TN, FN

**Run № 1**

45 епохи





Отново се наблюдава осцилиране в резултатите на точността, но то е значително по- малко.

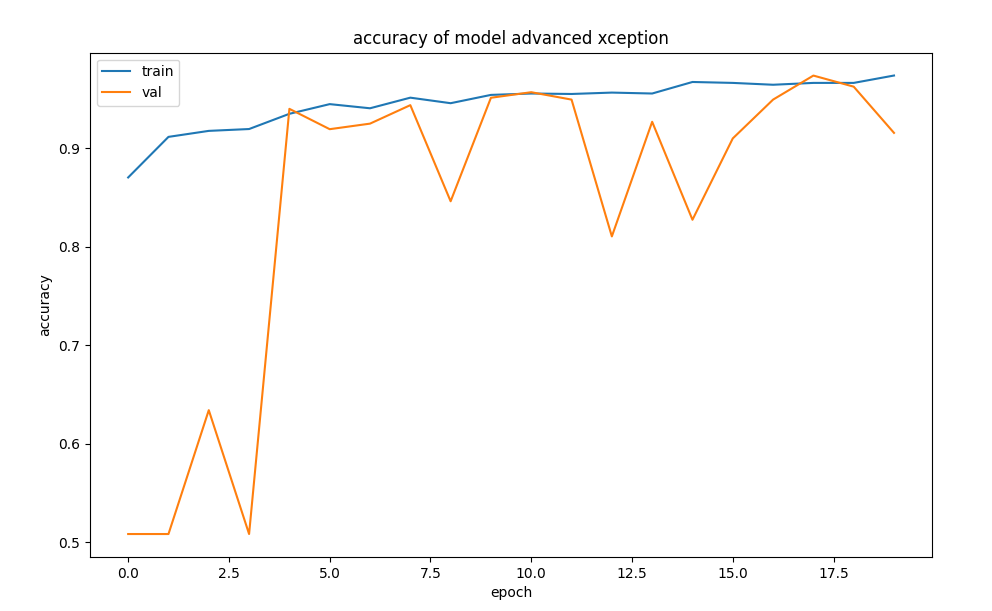
**Run № 2:**

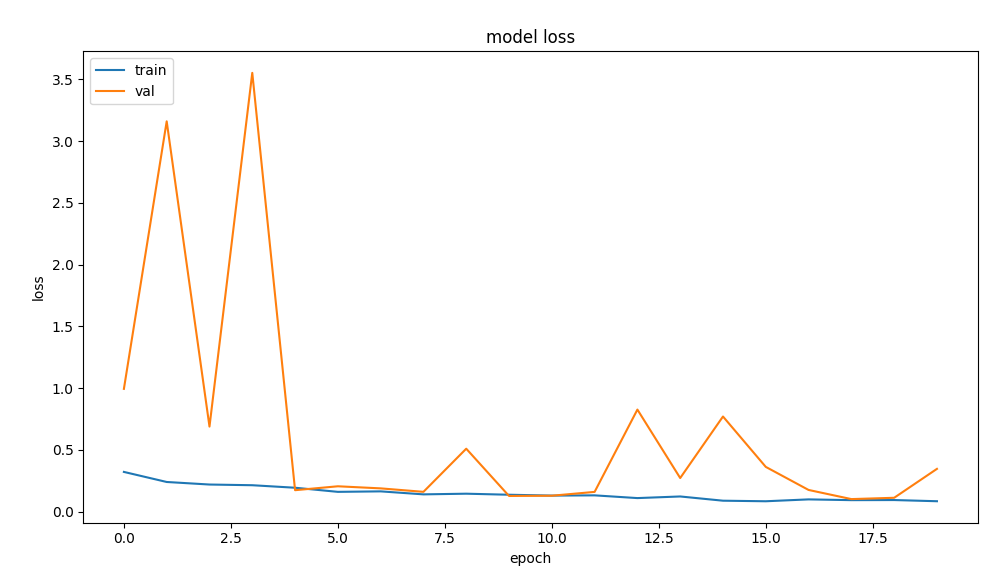
Използвана е същата архитектура на НМ, но с друг оптимизиращ алгоритъм- RMSProp:

model.compile(  
 optimizer=keras.optimizers.RMSprop(learning\_rate=0.0001),  
 loss="binary\_crossentropy",  
 metrics=["accuracy", "AUC",  
 keras.metrics.TruePositives(),  
 keras.metrics.FalsePositives(),  
 keras.metrics.TrueNegatives(),  
 keras.metrics.FalseNegatives()]  
)

Тренирането е извършено за 20 епохи.

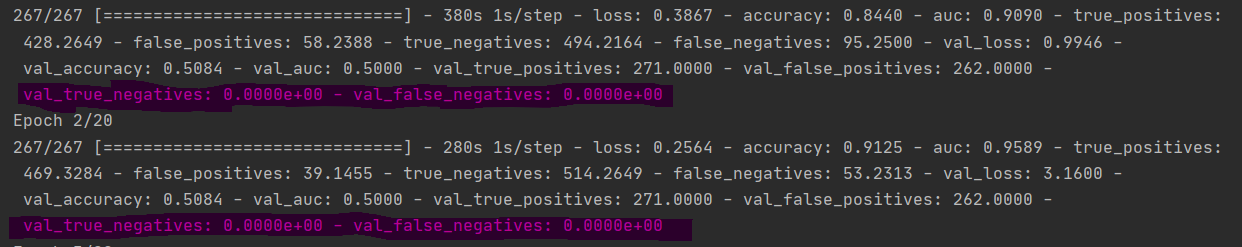
Резултати:



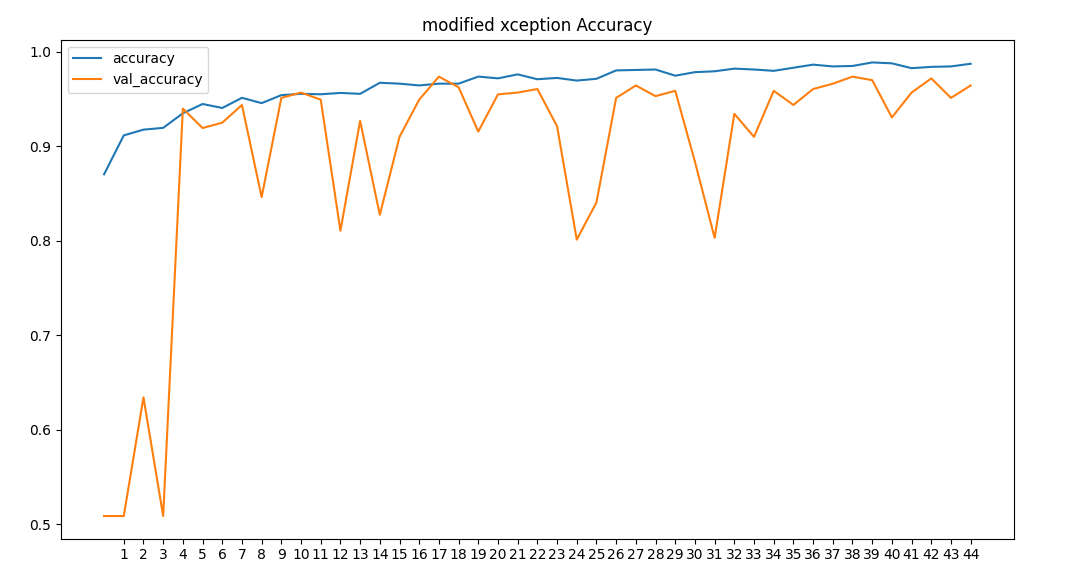


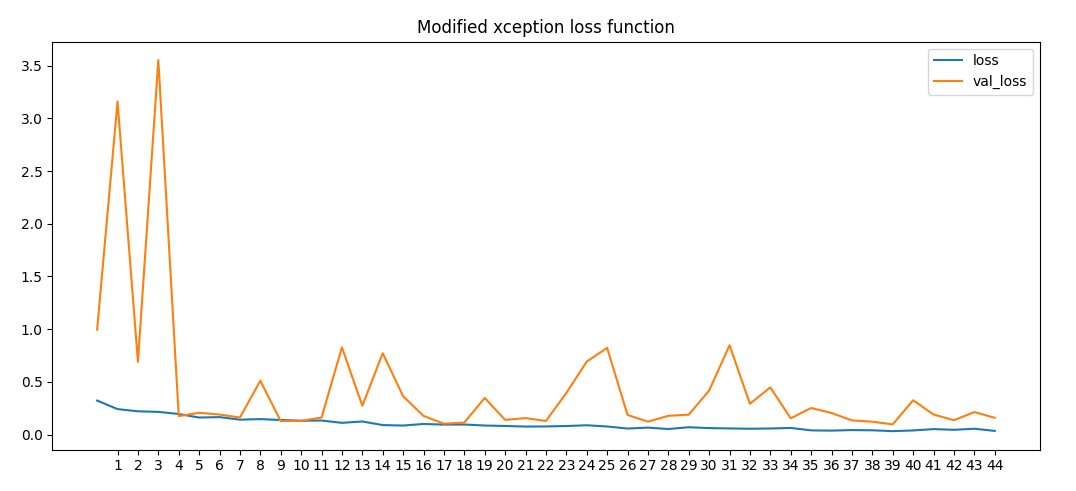
Осцилирането отново е по- малко спрямо Опит 1, но са нужни още данни(още трениране на мрежата), за да се направи заключение и да се открие ясна тенденция.

Интересен факт е, че в първите 2 епохи, невронната мрежа класифицира всички изображения към един единствен клас:



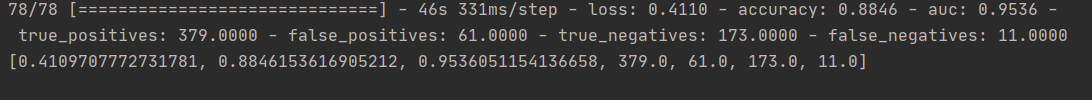
Следва трениране на мрежата за още 25 епохи(общо 45)





От графиките по-горе се наблюдава намаляване на осцилирането и засилване на монотонността след епоха 33. От графиката личи един потенциално най-добър(до момента) модел- от епоха 40.

За него след оценка чрез **тестовия** dataset(около 600 изображения) се получава:



От резултатите се вижда, че достигната точност **е 88.46%,** но стойността на loss функцията остава сравнително висока – 0.411.

Могат да бъдат изчислени и :

Чувствителността - True Positive Rate(TPR) = TP/ TP +FN= 0.972

Прецизността – Positive Predictive Value(PPV) = TP/TP+FP = 0.8614

В конкретния случай се разглеждат медицински изображения. Следователно, според мен e по-важно да се минимизира броя на False Negative класификациите(т.нар. Type 2 error), защото представлява по-голям риск.

**Опит 3.**

В този опит е използвана методиката „трансферирано обучение“(transfer learning). На база наблюдаваното осцилиране от предния опит, в текущия ще бъде използвана резолюция на изображенията- 128х128. Това позволява увеличаване на batch size параметъра(броят едновременно подавани изображения на мрежата) на 32.

**Run 1:**

Използвана е вградената в Tensorflow имплементация на невронната мрежа VGG16, в която са заредени теглата за ImageNet дейтасета. Тяхната промяна е забранена чрез trainable=false по време на трениране върху изображенията на белите дробове. Изображенията в този опит са заредени в RGB формат, защото използваната VGG имплементация го изисква. От архитектурата са махнати последните 3 слоя(пълносвързани и изходният слой с категориите) и са добавени нови, подходящи за текущо обработвания сет:

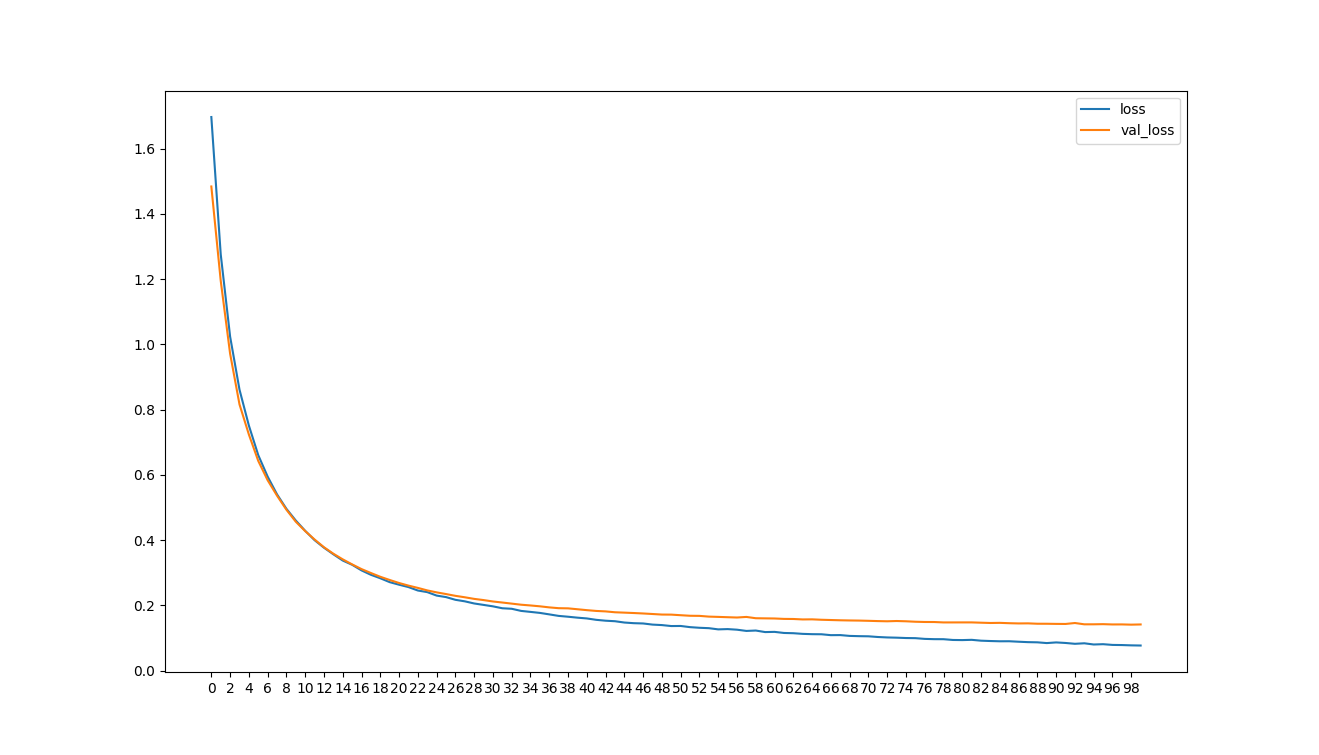
* GlobalAvaragePooling2D- пулинг слой
* Dense(1,activation=’sigmoid’)- изходен слой отразяващ резултата от мрежата в рамките на 1 неврон, чрез който се извършва класификацията. Като активираща функция е зададена сигмоидната, удобно ограничаваща резултата в интервала [0,1].

Извършено е трениране за 100 епохи, със следните параметри :

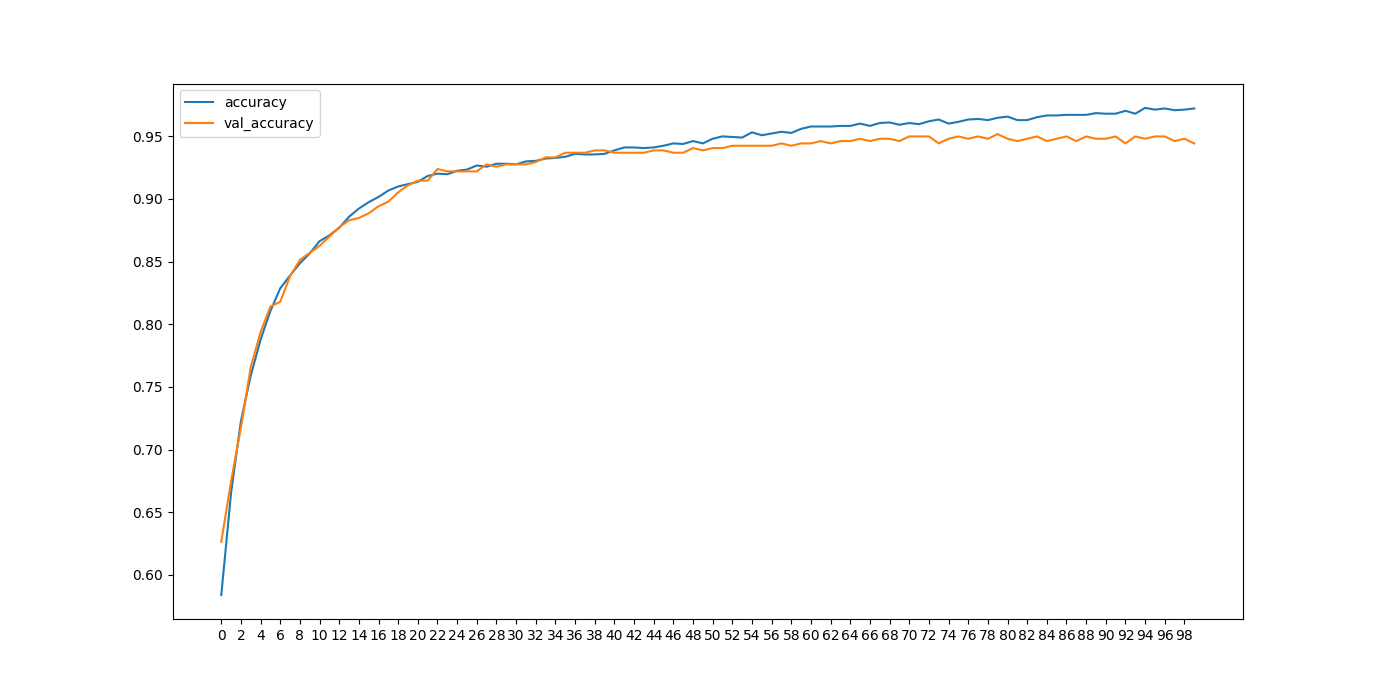
optimizer=keras.optimizers.Adam(learning\_rate=0.0001),  
loss="binary\_crossentropy"

При което се получават следните резултати:

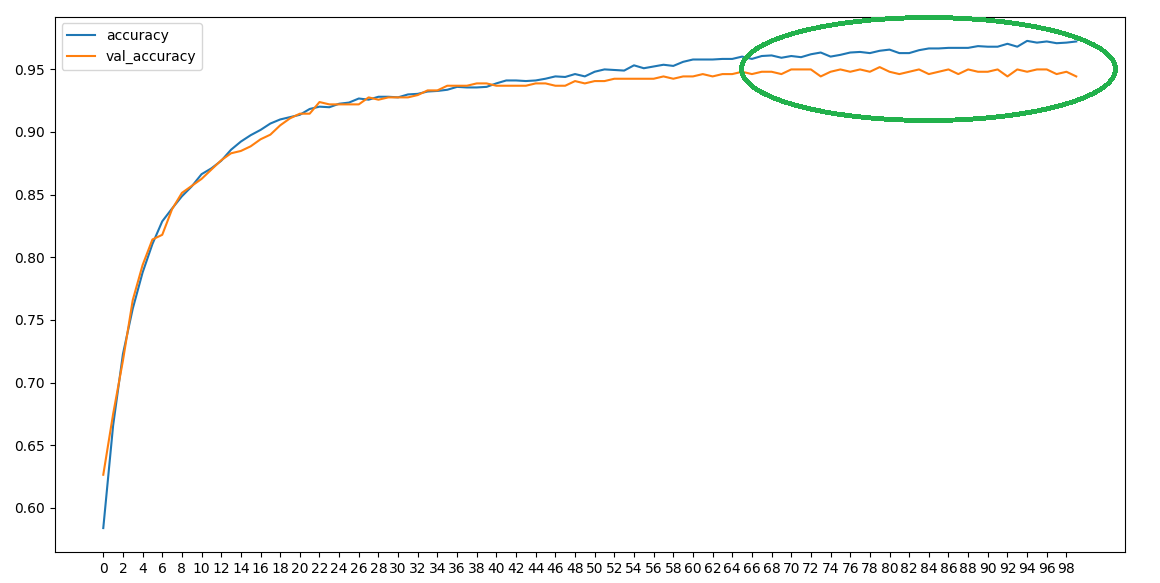
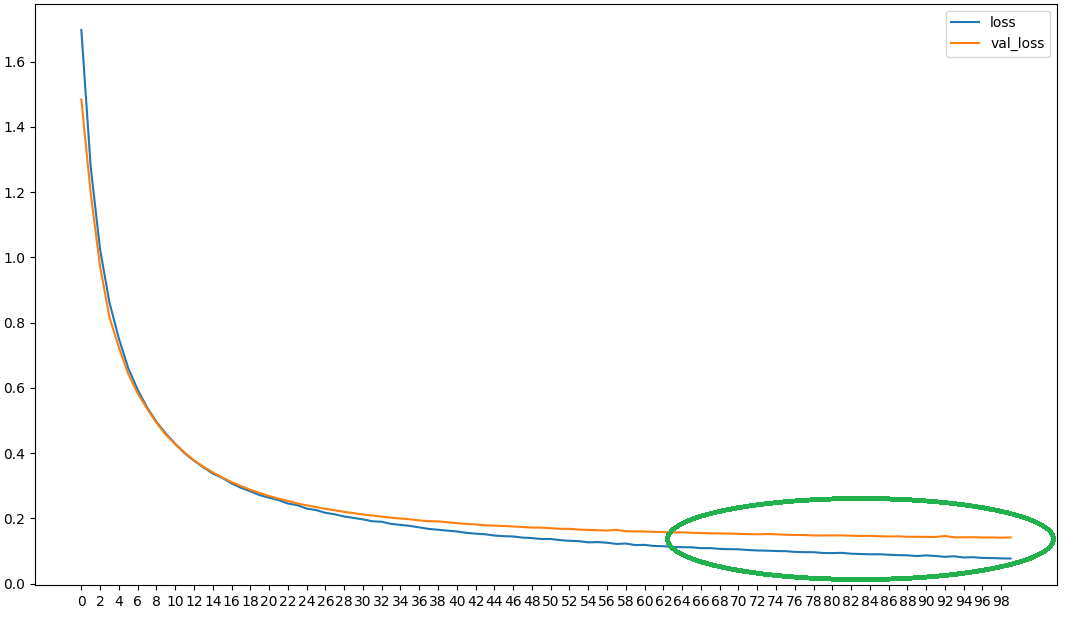
* За ценовата функция



* За точността



Наблюдават се много по-гладки криви на ценовата функция и точността, защото се изчисляват осреднените стойности за всеки batch от изображения, а той е с по-голям размер- 32, спрямо 8 в предишните опити. От графиките се вижда насищане стойностите за ценовата функция и точността след епоха 64-65- отбелязано със зелена елипса:



Може да се заключи, че стойностите на теглата в епоха №65 са оптимални за текущата архитектура. В следващите епохи се вижда повишаване на точността при тренирането, но насищане(на места и спад) във валидационната точност- индикация за пренагаждане на модела спрямо данните за трениране. Overfitting-ът в случая може да се определи като слаб, защото разликата между валидационната и трениращата точност не е голяма- от порядъка на 5%. По-нататъшно трениране(след епоха 100) няма да доведе до по-добри резултати, а напротив- моделът няма да може да генерализира достатъчно добре и да класифицира правилно нови, невиждани до сега данни.

**Run №2**

В този опит към архитектурата на VGG16 от предния опит ще бъдат добавени следните модификации:

* **Data Augmentation слой-** произволни модификации върху входните изображения
* **Dropout слой** с параметър вероятност 0.3 – за да се избегне пренагаждането на модела спрямо данните за трениране и да успява да генерализира по- добре
* **Early stopping-** използване на техниката за навременно спиране на тренирането за да се избегне **overfitting.** Като критерий за прекратяване на обучението е зададен – **липса на промяна в стойността на валидационната ценова функция в 3 последователни епохи.**

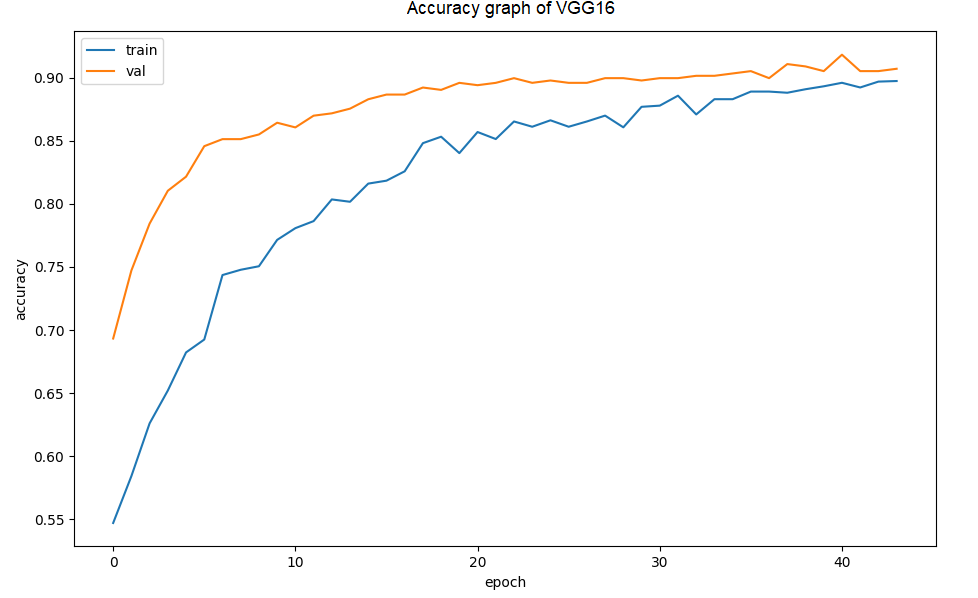
Код на създаване на модела:

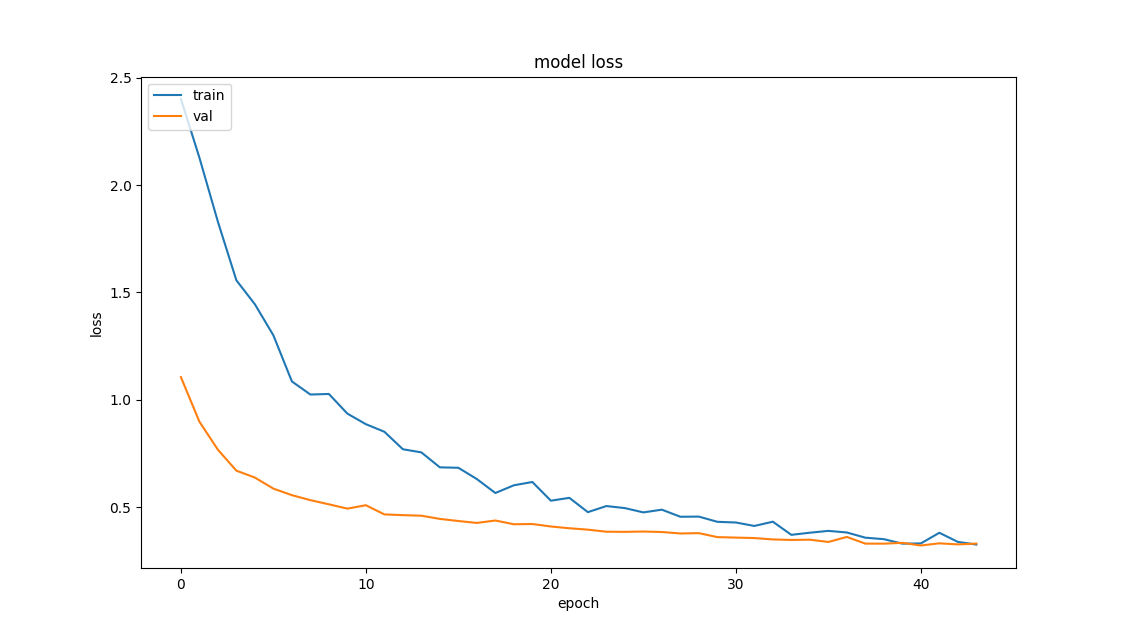
def model\_create\_vgg16(input\_shape):  
 base\_mdl = krs.applications.vgg16.VGG16(include\_top=False, Input\_shape=input\_shape, weights='imagenet')  
 base\_mdl.trainable = False  
 inputs = krs.Input(input\_shape)  
 inp = data\_augm(inputs)  
 inp = krs.applications.vgg16.preprocess\_input(inp)  
 val = base\_mdl(inp, training=False)  
 val = krs.layers.GlobalAveragePooling2D()(val)  
 val= krs.layers.Dropout(0.3)(val)  
 outputs = krs.layers.Dense(1, activation='sigmoid')(val)  
  
 return krs.Model(inputs, outputs)

Код на компилация и трениране на модела:

def train\_vgg():  
 training = datalib.load\_dataset(TRAINING\_PATH)  
 val\_ds = datalib.load\_validation(TRAINING\_PATH)  
 model = mdl.model\_create\_vgg16(input\_shape=(128, 128, 3))  
 epochs = 50  
 callbacks = [  
 keras.callbacks.ModelCheckpoint("vgg2/save\_at\_{epoch}.h5", save\_best\_only=True),  
 # keras.callbacks.CSVLogger(filename="xception\_log.csv", separator=',', append=True)  
 keras.callbacks.EarlyStopping(monitor='val\_loss', patience=3, mode='auto')  
 ]  
 model.compile(  
 optimizer=keras.optimizers.Adam(learning\_rate=0.0001),  
 loss="binary\_crossentropy",  
 metrics=["accuracy", "AUC",  
 keras.metrics.TruePositives(),  
 keras.metrics.FalsePositives(),  
 keras.metrics.TrueNegatives(),  
 keras.metrics.FalseNegatives()]  
 )  
 history = model.fit(  
 training, epochs=epochs, callbacks=callbacks, validation\_data=val\_ds,  
 )

Резултати от тренирането :





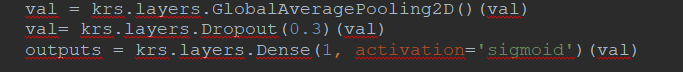
От представените графики на резултатите за стойността на ценовата функция и точността се вижда, че техниката за ранно спиране на тренирането е прекратила процеса в епоха 43(не е имало подобрение в епохите 41,42,43). Наблюдава се и минимално осцилиране, което най-вероятно се дължи на добавеното обогатяване на данните, внасящо по-голямо разнообразие в изображенията и респективно „случайност“. Интерес буди и фактът, че валидационната точност е по- висока от точността на трениране, а стойността на ценовата функция при валидацията е по-ниска от тази при трениране(т.е. моделът се справя по- добре при валидацията от колкото при тренирането). Това е така, защото архитектурата притежава изходен Dropout слой с определена вероятност, който е активен **само по време на тренирането.** Изкуствено затрудняваме мрежата по време на тренирането(загубата на информация от неврони), затова е логично и точността/ценовата функция да имат по-ниска/висока стойност.

Оптималните тегла при тренирането са достигнати в епоха 41, следователно тя ще бъде използвана и при оценка с тестовия дейтасет.

Тестване:

**Run №3**

В предишния run бяха премахнати пълно свързаните крайни слоеве на традиционната VGG16 архитектура и бяха добавени GlobalAvgPooling слой и веднага след него – класификатора със сигмоидната функция:



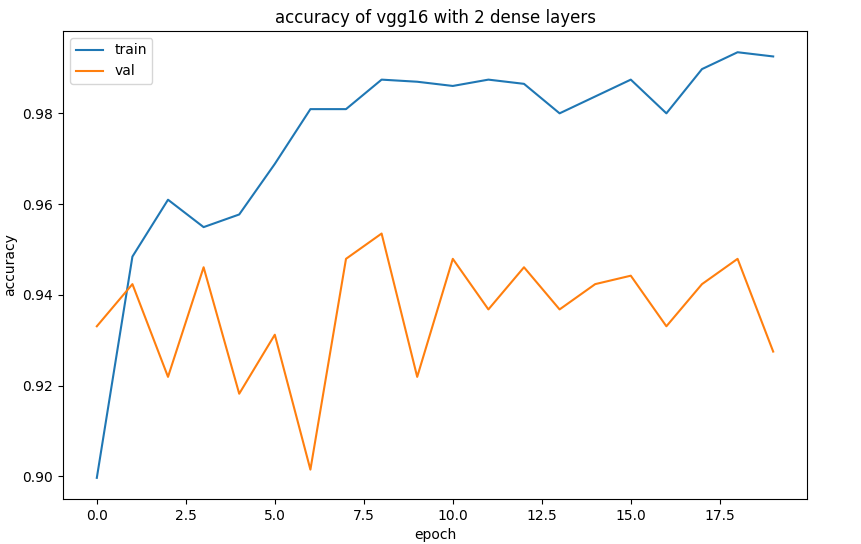
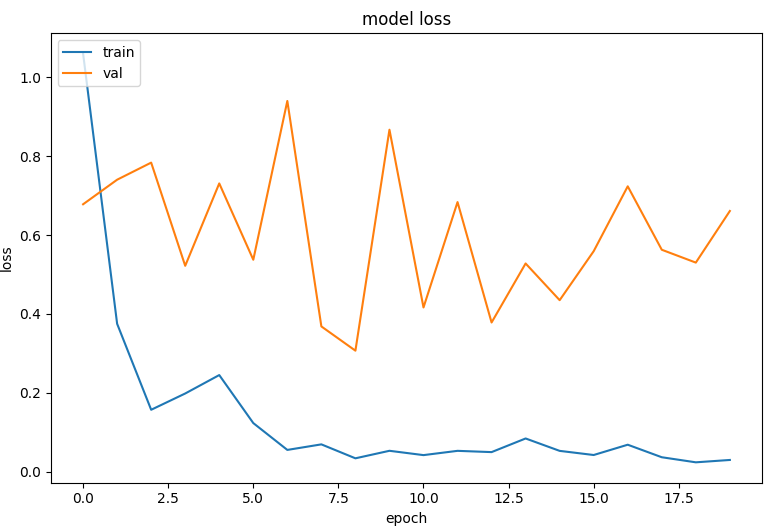
Както се вижда от графиките на предното пускане, точността, която се достига при трениране/валидация е от порядъка на 80-90%, а тази при тестване с тестовия дейтасет- дори по-ниска. Много вероятно е изходните слоеве да не успяват да усвоят достатъчно добре специфичността на данните. Затова ще бъде направена промяна в крайните слоеве на мрежата- flatten слой и два плътно свързани слоя с ReLu активационна функция преди същинския класификатор(маркирани в син цвят). Така архитектурата придобива следния вид:

def model\_create\_vgg16(input\_shape):  
 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.Flatten()(val)  
 val = krs.layers.Dense(2048, activation="relu")(val)  
 val = krs.layers.Dense(2048, activation="relu")(val)  
 outputs = krs.layers.Dense(1, activation='sigmoid')(val)  
 return krs.Model(inputs, outputs)

За първоначалното трениране няма да бъде ползван dropout слой. Параметри на тренирането :

* Използван оптимизационен метод: Adam
* Брой епохи: 20
* Learning rate: 0.0001
* Ценова функция: binary crossentropy

**Резултати**

****

В графиките на резултатите отново се наблюдава леко осцилиране(малко по-силно изразено във валидационната част), но доста по-високи нива на достигната точност спрямо предишния опит. Въпреки това, отново се вижда разлика между валидационна точност и точност за трениране от порядъка на 6-7%, която предразполага използването на някакъв вид регуляризация(например въвеждането на dropout слой).

Използването на по-голям batch size(32) чувствително успя да намали осцилирането, но не и да го премахне. Друга възможна причина за осцилирането може да бъде learning rate хиперпараметърът - да има прекалено висока стойност за текущия разглеждан проблем и да „прескача“ оптималното решение. За справяне с това явление в следващото трениране ще бъде използвана и техниката **learning rate schedule,** при която LR параметърът ще бъде намаляван прогресивно във времето. Това ще спомогне по-лесното намиране на оптималното решение, защото теглата ще се обновяват с по- малки стъпки. Това, разбира се, крие и риск- възможно е засядане в локален минимум, ако LR параметърът е прекалено малък.