**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:

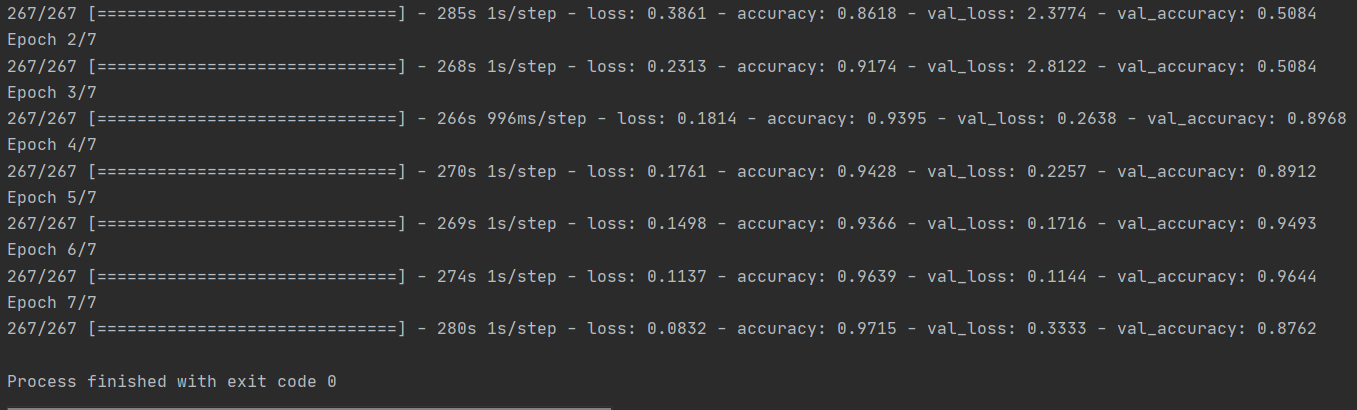
def model\_create\_xception(input\_shape, num\_classes):  
 inputs = krs.Input(shape=input\_shape)  
  
 # Entry block  
 x = krs.layers.experimental.preprocessing.Rescaling(1.0 / 255)(inputs)  
 x = krs.layers.Conv2D(32, 3, strides=2, padding="same")(x)  
 x = krs.layers.BatchNormalization()(x)  
 x = 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:**

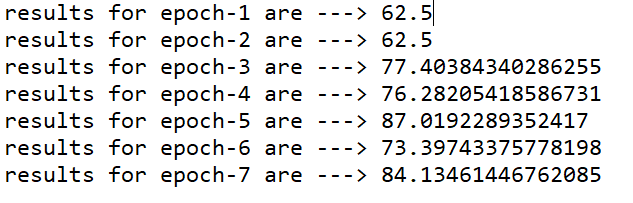
Параметри:

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

Резултати при тренирането:



Резултати при тестването:



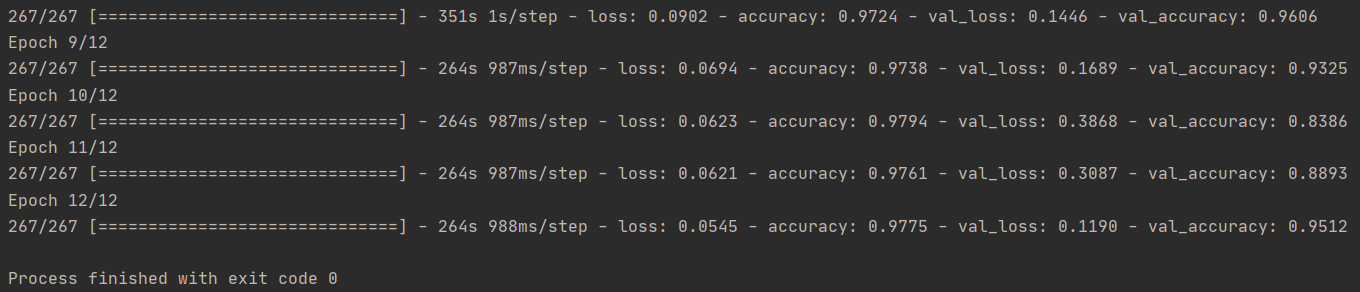
**Run № 2:**

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

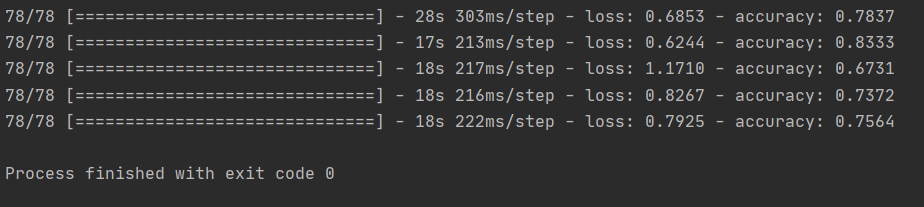
Параметри:

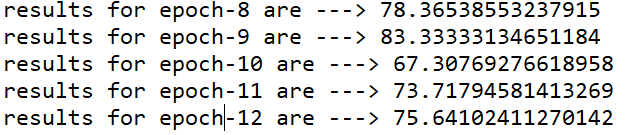
optimizer- Adam(learning\_rate=0.001) ; loss function- binary crossentropy ; **number of epochs – 5(total of 12)**

Резултати при тренирането:

****

Резултати при тестването:



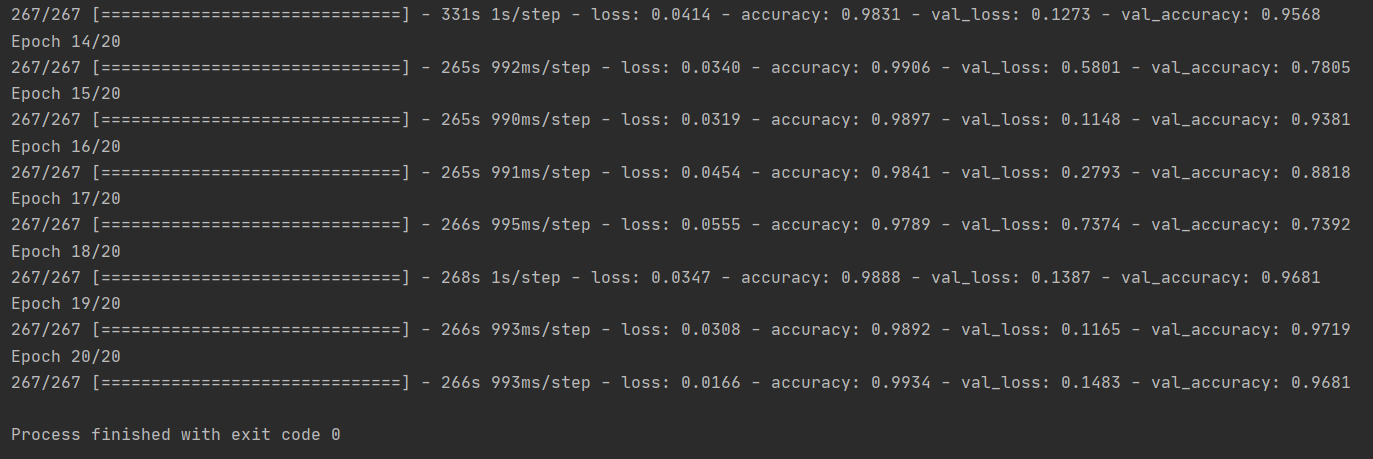


**Run № 3:**

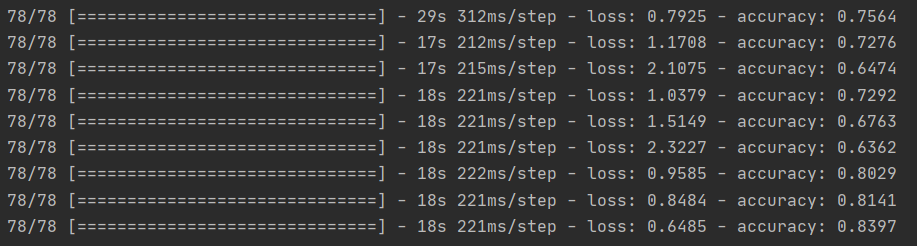
Зареждане на модела от .h5 файл в състояние мрежата- епоха 12 и трениране за още 8 епохи

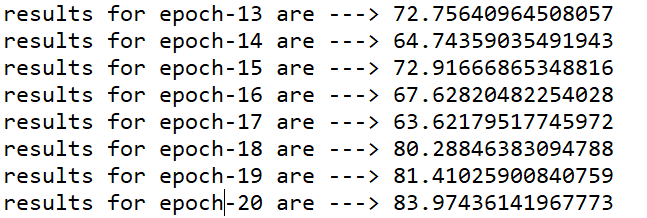
Параметри:

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



Резултати от тестването:



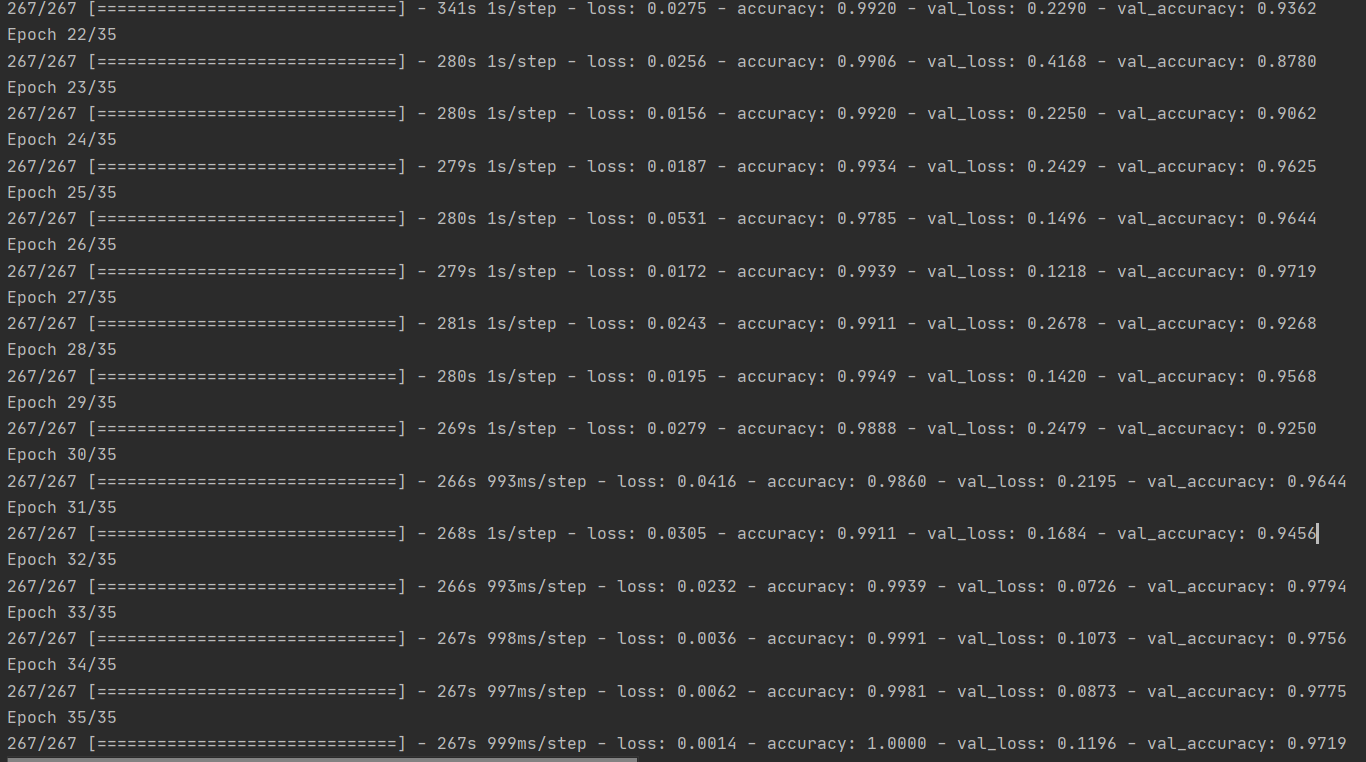


**Run № 4:**

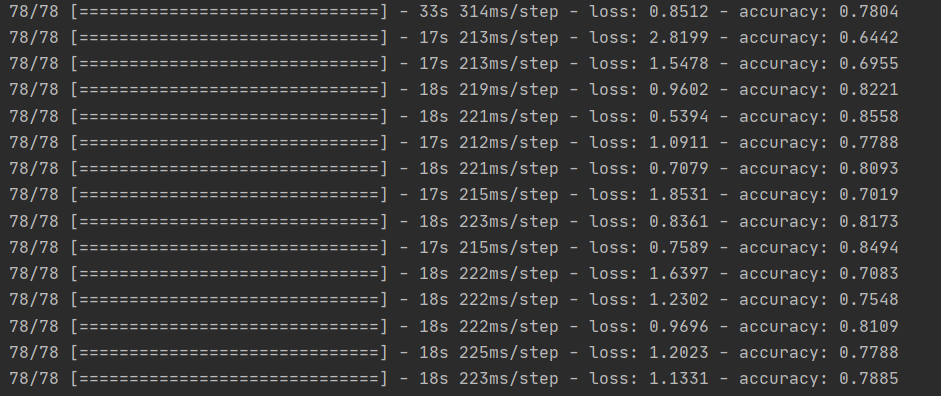
Зареждане на модела от .h5 файл в състояние мрежата- епоха 20 и трениране за още 15 епохи

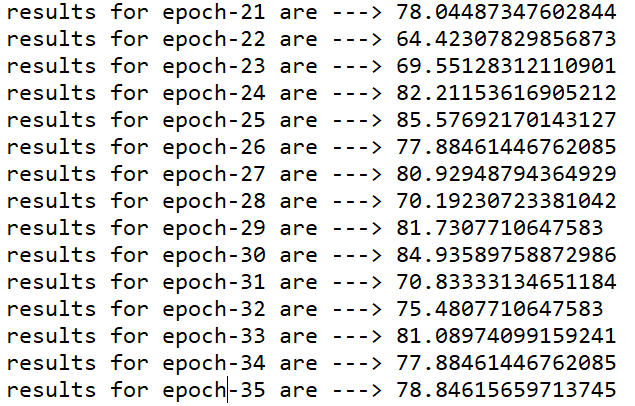
Параметри:

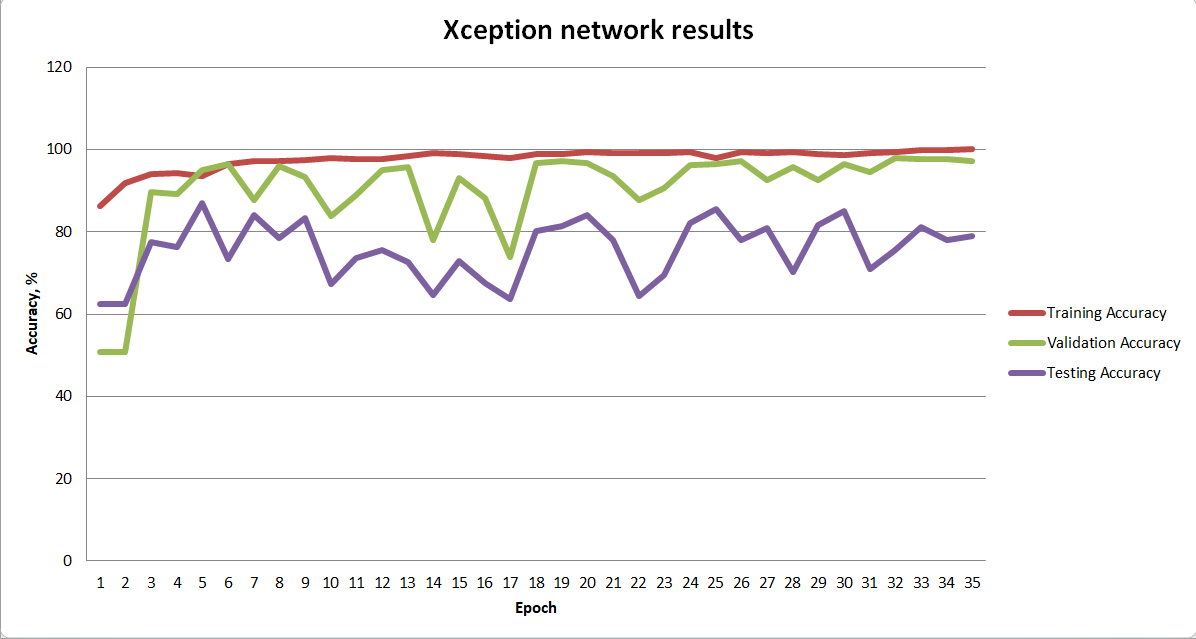
optimizer- Adam(learning\_rate=0.001) ; loss function- binary crossentropy ; **number of epochs – 15(total of 35)**

****

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

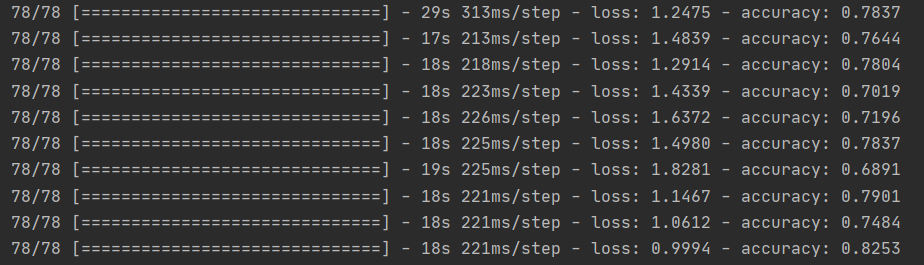
Параметри:

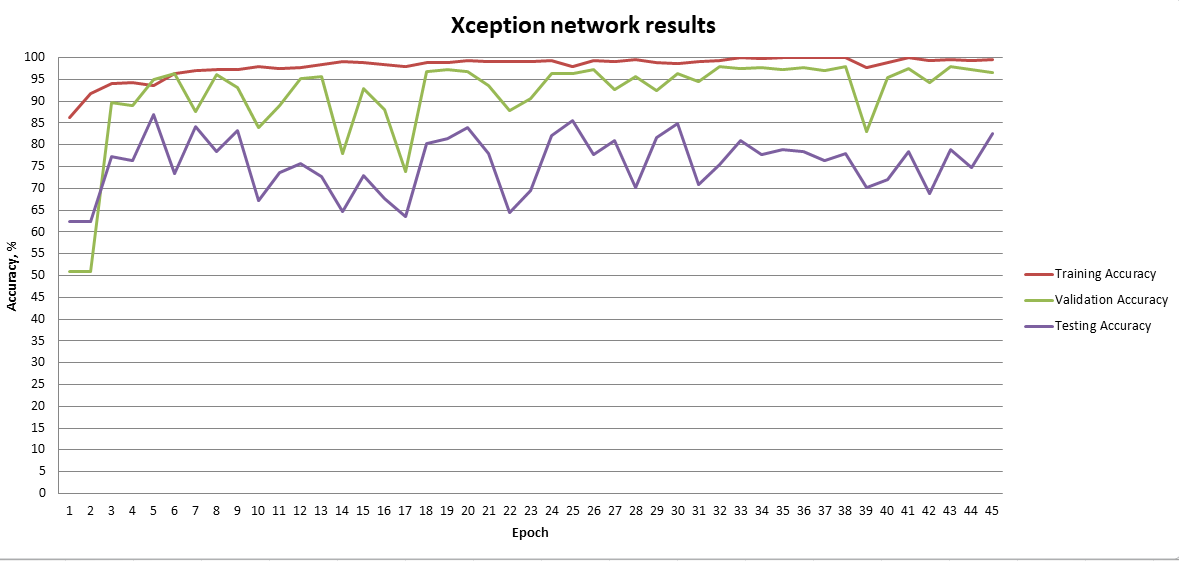
optimizer- Adam(learning\_rate=0.001) ; loss function- binary crossentropy ; **number of epochs – 10(total of 45)**

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

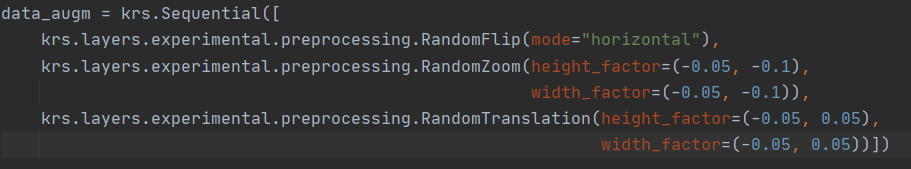
3. Промяна в архитектурата на мрежата.

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

**Опит 2:**

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

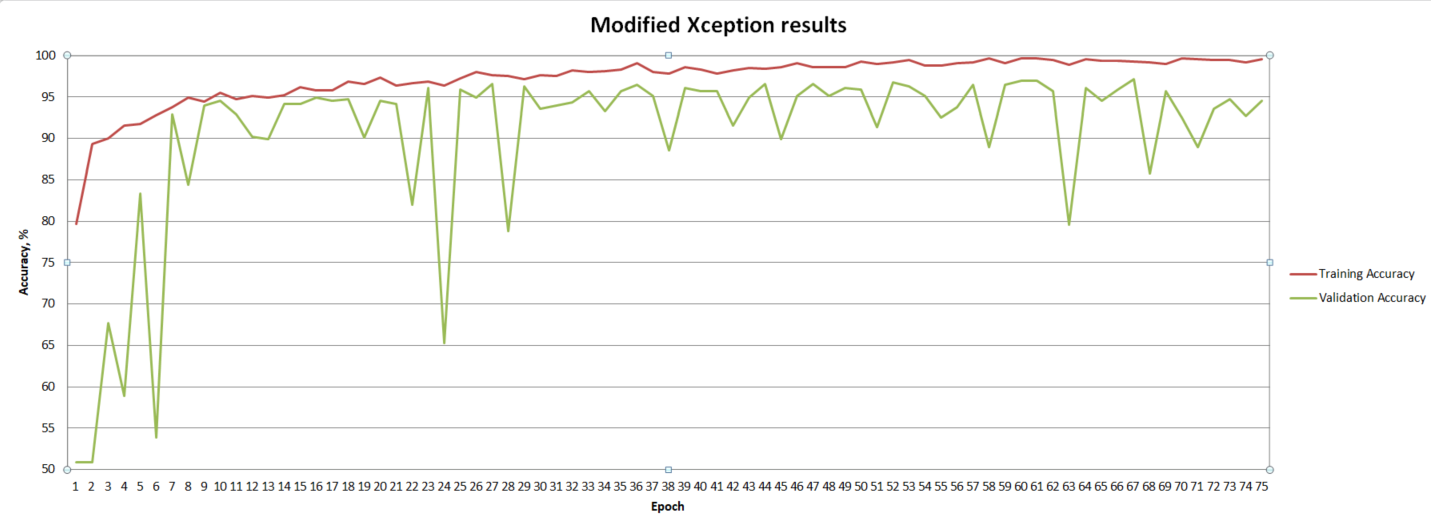
* Добавяне на data augmentation **към архитектурата на невронната мрежа(**обогатяването се извършва „on-line“ върху GPU като слоеве от мрежата**)**

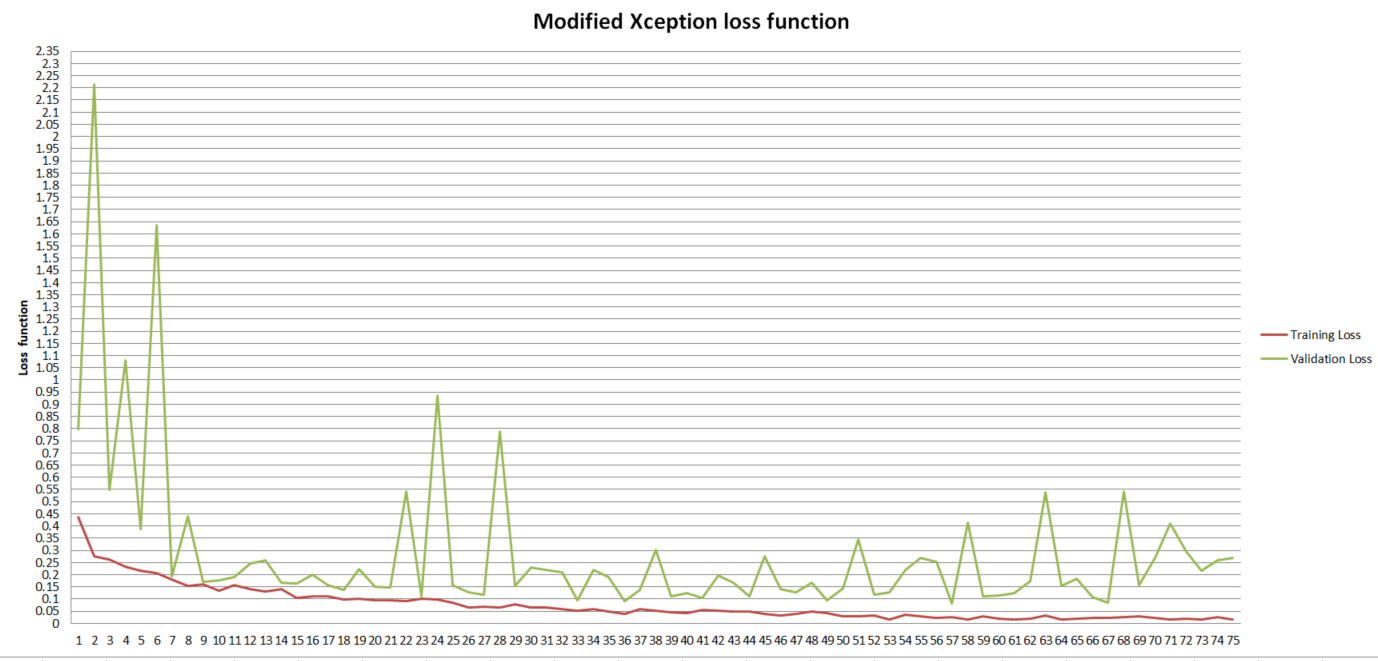


* Промяна на стойността на вероятността на DropOut слоя в мрежата на 0.6 (предишна стойност – 0.5)
* Промяна на хиперпараметъра learning rate на оптимизационния метод Adam на 0.00005(предишна стойност 0.001)
* Стойността на batch size е запазена същата- batch size= 8
* Зададено е отпечатване на следните 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.

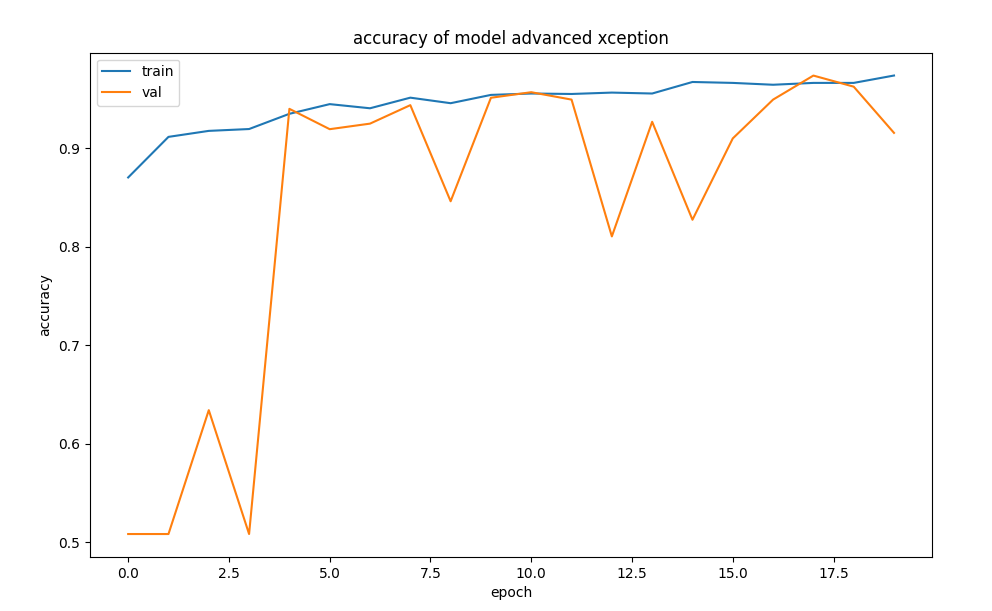
**Run № 2:**

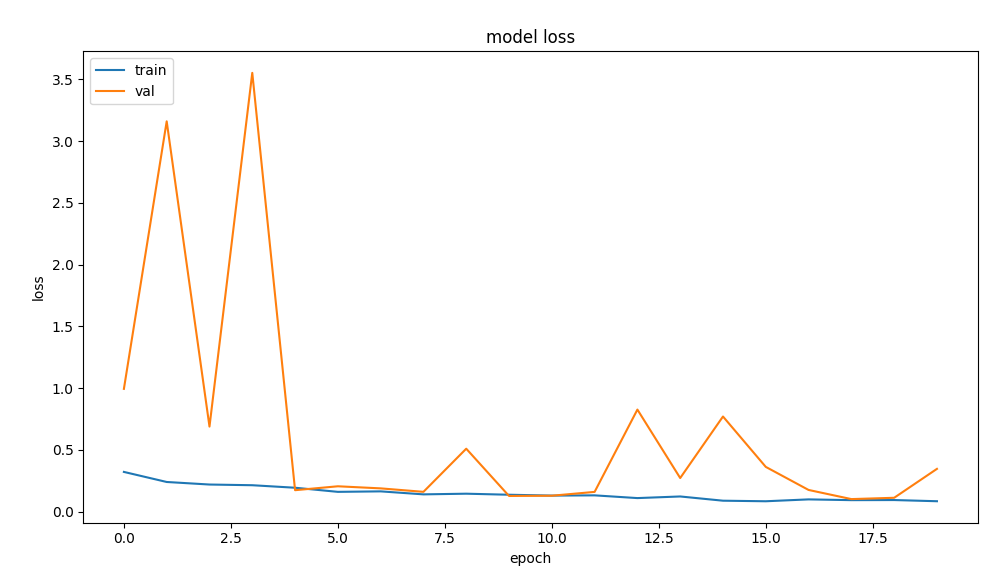
Използвана е същата архитектура на НМ, но с друг оптимизиращ алгоритъм- RMSProp:

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.

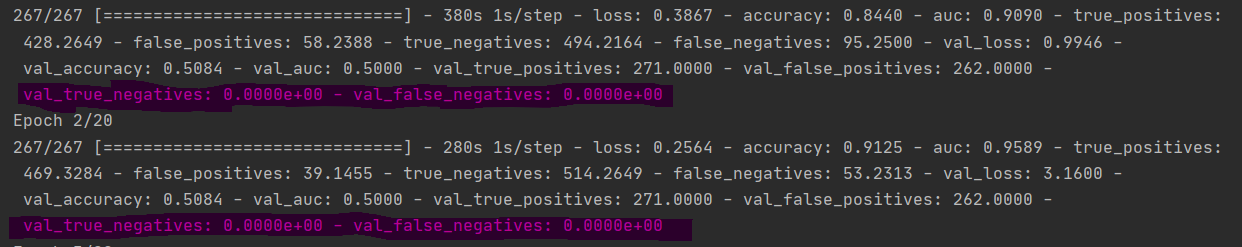
Резултати:



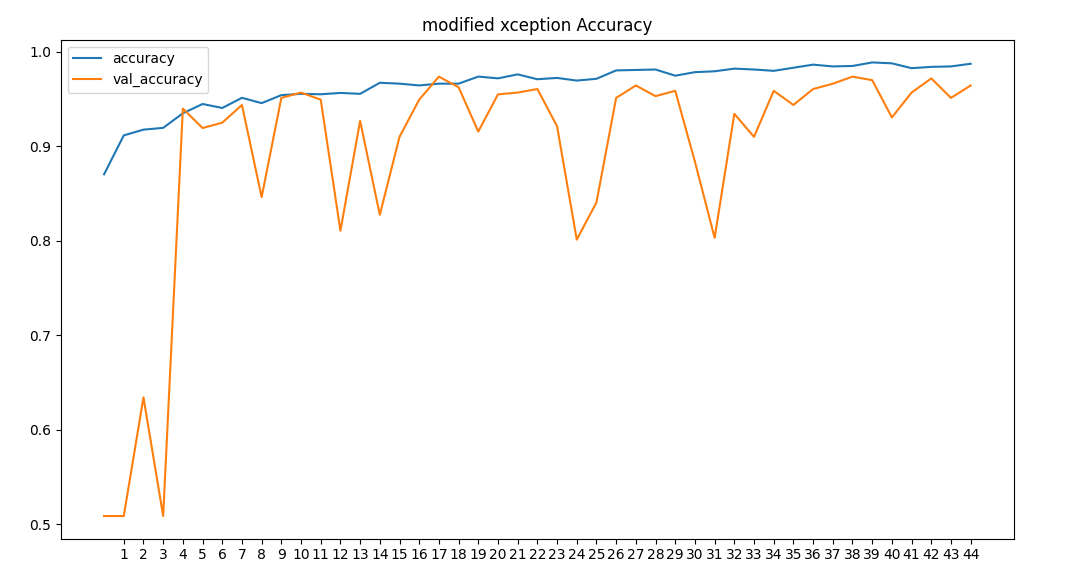


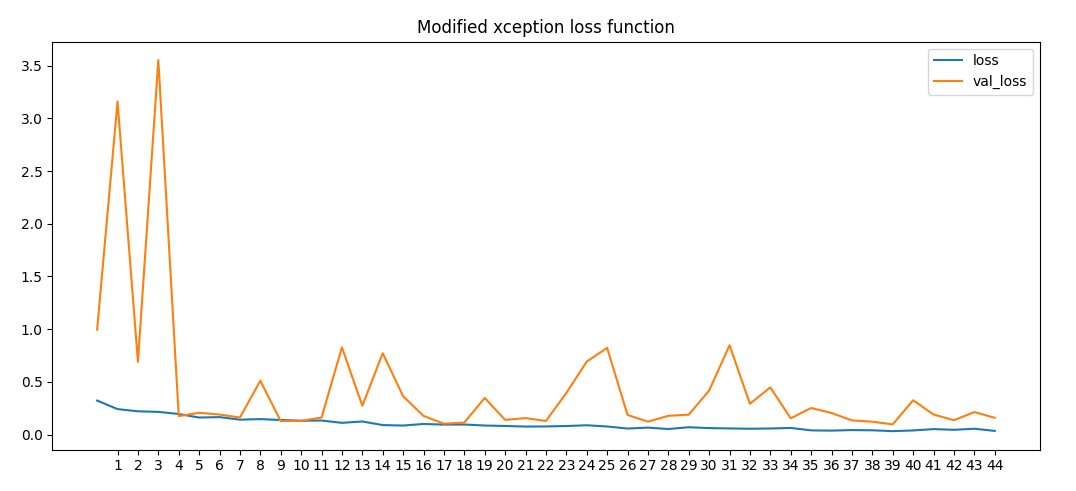
Осцилирането отново е по- малко спрямо Опит 1, но са нужни още данни(още трениране на мрежата), за да се направи заключение и да се открие ясна тенденция.

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



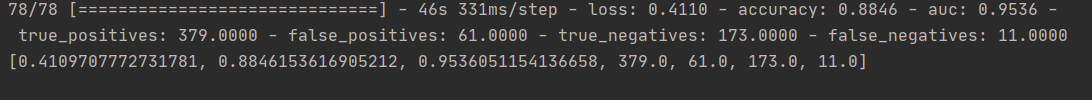
Следва трениране на мрежата за още 25 епохи(общо 45)





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).

**Run 1:**

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