Multi Cancer Identification and Segmentation Report

Youssef Amr Abdelmonem

Lojain Wail Mohamed

Mohamed Galal Mohamed

Mohamed Hassan Mahmoud

Nadine Walid Adly

1. Data preparation:

- Preprocessing:

- Up sampling in classification of:

  1) Brain and breast.

  2) Normal, Malignant or breast.

  Snapshot Before and after Up sampling

```
[32] ## checking for imbalance
     print("Breast - Normal: ",len(normal_train))
     print("Breast - Benign: ",len(benign_train))
     print("Breast - Malignant: ",len(malignant_train))
```

```
Breast - Normal:  103
Breast - Benign:  397
Breast - Malignant:  180
```

High Imbalance in breast data -> Trying upsampling

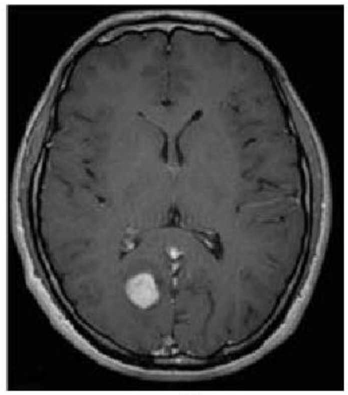
```
[33] from sklearn.utils import resample
     normal_train = resample(normal_train,
                                  replace=True,
                                  n_samples=397)
     malignant_train = resample(malignant_train,
                                  replace=True,
                                  n_samples=397)
```

```
[34] ## after upsampling
     print("Breast - Normal: ",len(normal_train))
     print("Breast - Benign: ",len(benign_train))
     print("Breast - Malignant: ",len(malignant_train))
```

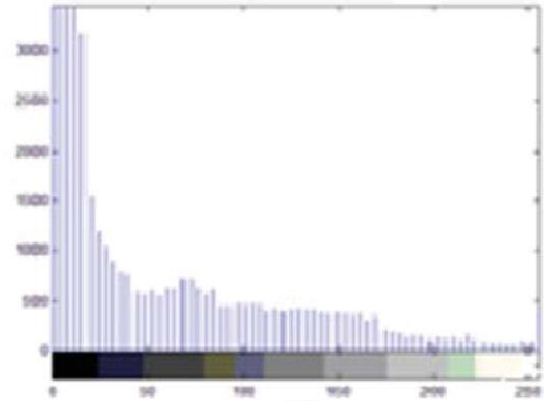
```
Breast - Normal:  397
Breast - Benign:  397
Breast - Malignant:  397
```

- # Enhancement:

  - Histogram Equalization.

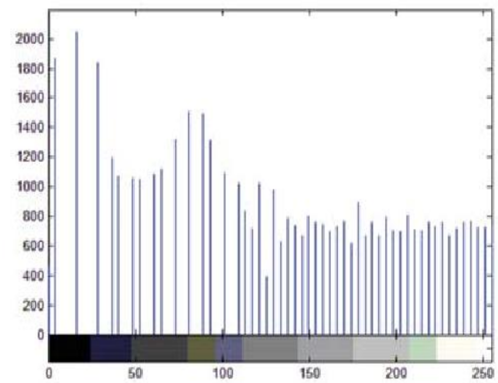


(a)



(b)

(c)



(d)

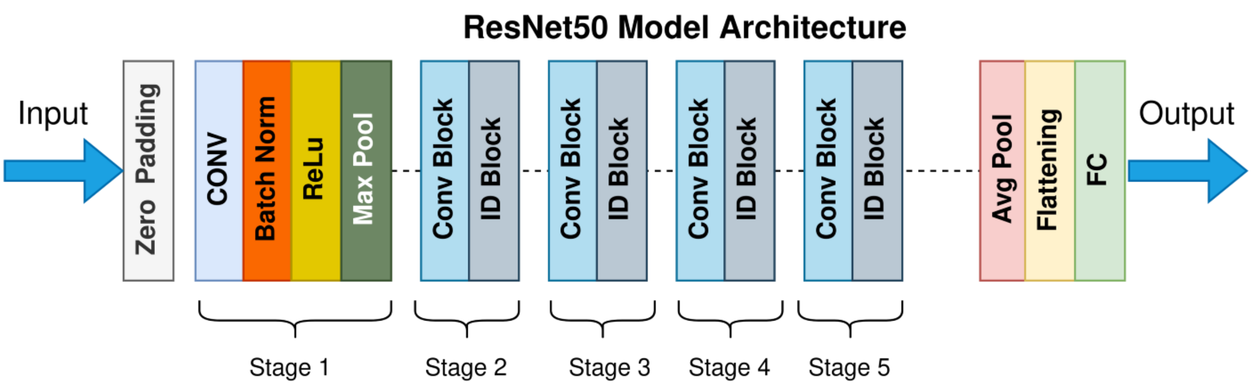
- Sharpening.
- Median filter to reduce noise in images (We tried it but didn't make much enhancement).

Note: None of them enhanced our classification.

- Image Processing:
    - We resized images to 160 × 160 × 3
    - Image Scaling (Divide images by 255 to Normalize)

2. Models used:

- First level classification (Brain and Breast):
  - **ResNet50 Pretrained In ImageNet (BestOne)**
  - The 50-layer ResNet uses a bottleneck design for the building block. A bottleneck residual block uses 1×1 convolutions, known as a "bottleneck", which reduces the number of parameters and matrix multiplications. This enables much faster training of each layer.



Snapshot from ResNet50 code:

```python
# Model - ResNet50
from keras.applications import ResNet50
from keras import layers
from keras.optimizers import Adam, RMSprop
from keras.callbacks import EarlyStopping

base_Neural_Net= ResNet50(input_shape=(IMG_SIZE,IMG_SIZE,3), weights='imagenet', include_top=False)
model=Sequential()

model.add(base_Neural_Net)
model.add(Dropout(0.2))
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(256,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(1,activation='sigmoid'))

for layer in base_Neural_Net.layers:
    layer.trainable = False


model.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy' , tf.keras.metrics.Recall(),tf.keras.metrics.Precision()]
)

es = EarlyStopping(
    monitor='val_accuracy',
    mode='max'
)
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2,
                              patience=3, min_lr=0.00001)
history = model.fit(
    X_Brain_Train, Y_Brain_Train,
    steps_per_epoch=50,
    epochs=30,
    validation_split=0.2,verbose=1,callbacks=[reduce_lr]
)


score = model.evaluate(X_Brain_Test, Y_Brain_Test, verbose=1)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}, Recall: {score[2]}, Precision: {score[3]}')
model.summary()
```

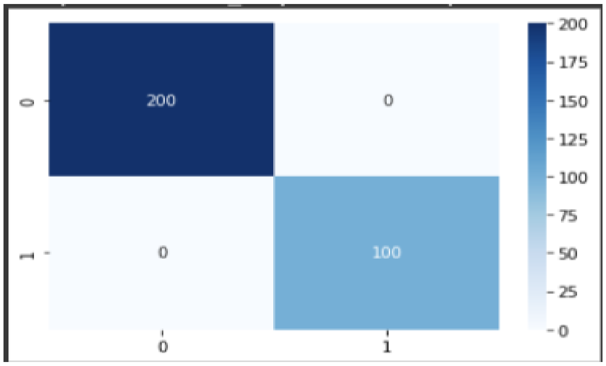
Results:

As Shown in Figures Train Validation results is great and also test which leads to test accuracy is 100%

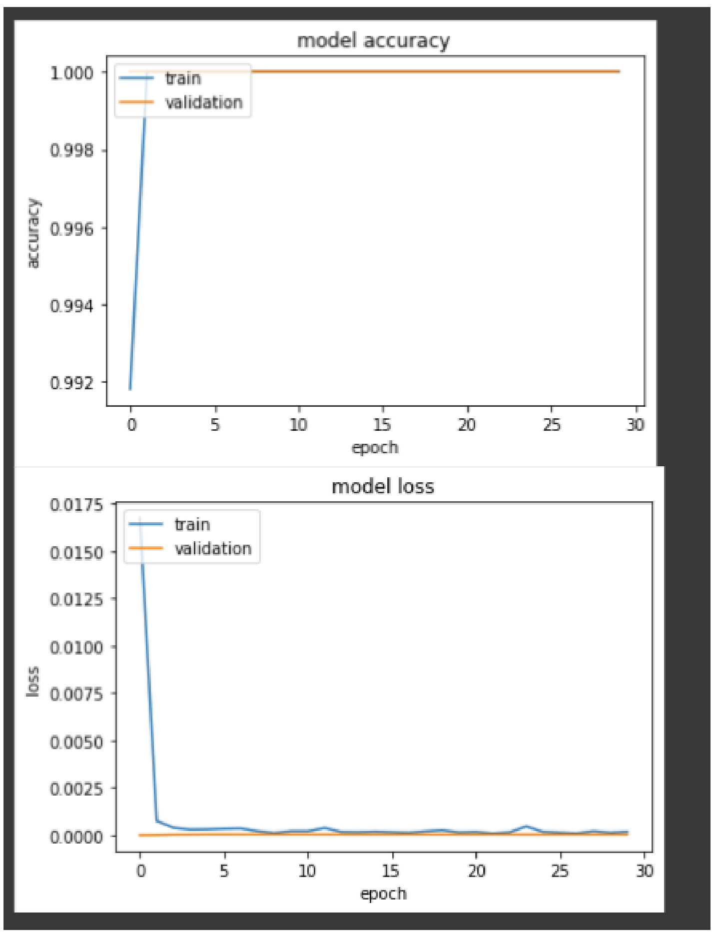
```
Epoch 25/30
50/50 [==============================] - 6s 111ms/step - loss: 1.8130e-04 - accuracy: 1.0000 - recall: 1.0000 - precision: 1.0000 - val_loss: 3.6437e-05 - val_accuracy: 1.0000 - val_recall: 1.0000 - val_precision: 1.0000 - lr: 1.0000e-05
Epoch 26/30
50/50 [==============================] - 6s 112ms/step - loss: 1.3728e-04 - accuracy: 1.0000 - recall: 1.0000 - precision: 1.0000 - val_loss: 3.6717e-05 - val_accuracy: 1.0000 - val_recall: 1.0000 - val_precision: 1.0000 - lr: 1.0000e-05
Epoch 27/30
50/50 [==============================] - 6s 119ms/step - loss: 9.9708e-05 - accuracy: 1.0000 - recall: 1.0000 - precision: 1.0000 - val_loss: 3.7376e-05 - val_accuracy: 1.0000 - val_recall: 1.0000 - val_precision: 1.0000 - lr: 1.0000e-05
Epoch 28/30
50/50 [==============================] - 6s 126ms/step - loss: 2.1469e-04 - accuracy: 1.0000 - recall: 1.0000 - precision: 1.0000 - val_loss: 3.7403e-05 - val_accuracy: 1.0000 - val_recall: 1.0000 - val_precision: 1.0000 - lr: 1.0000e-05
Epoch 29/30
50/50 [==============================] - 6s 112ms/step - loss: 1.3433e-04 - accuracy: 1.0000 - recall: 1.0000 - precision: 1.0000 - val_loss: 3.6583e-05 - val_accuracy: 1.0000 - val_recall: 1.0000 - val_precision: 1.0000 - lr: 1.0000e-05
Epoch 30/30
50/50 [==============================] - 6s 112ms/step - loss: 1.8624e-04 - accuracy: 1.0000 - recall: 1.0000 - precision: 1.0000 - val_loss: 3.5949e-05 - val_accuracy: 1.0000 - val_recall: 1.0000 - val_precision: 1.0000 - lr: 1.0000e-05
10/10 [==============================] - 2s 159ms/step - loss: 4.7765e-05 - accuracy: 1.0000 - recall: 1.0000 - precision: 1.0000
```

Test loss	Test accuracy	Recall	Precision
0.000048	1.000000	1.000000	1.000000

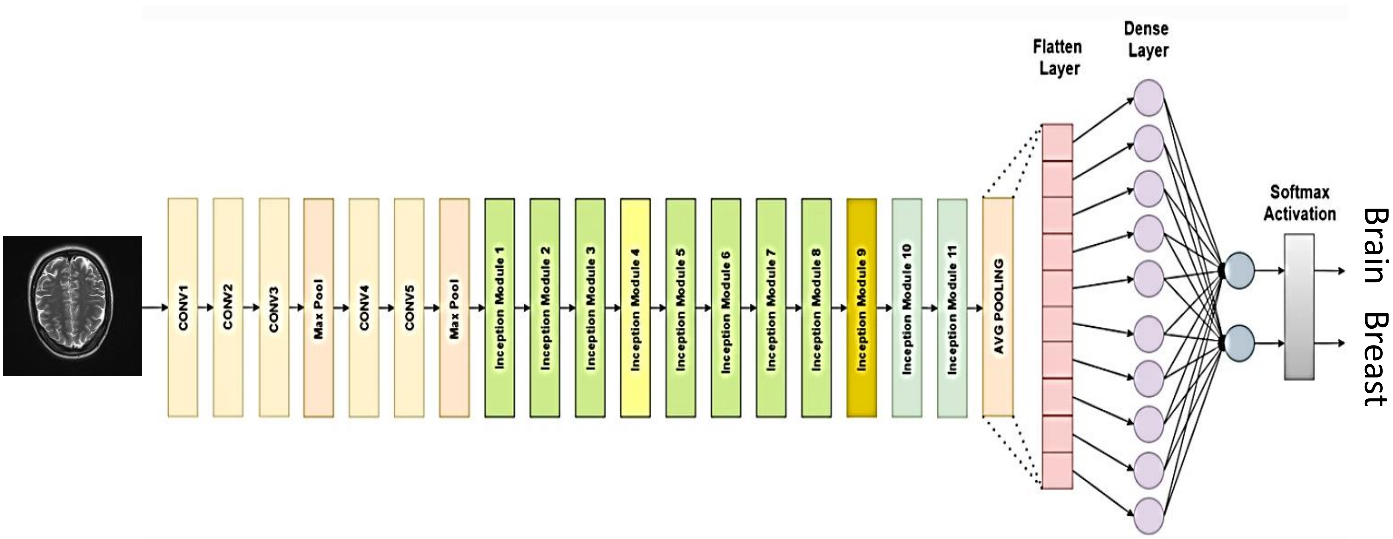
Confusion Matrix



Train vs Validation in accuracy and loss

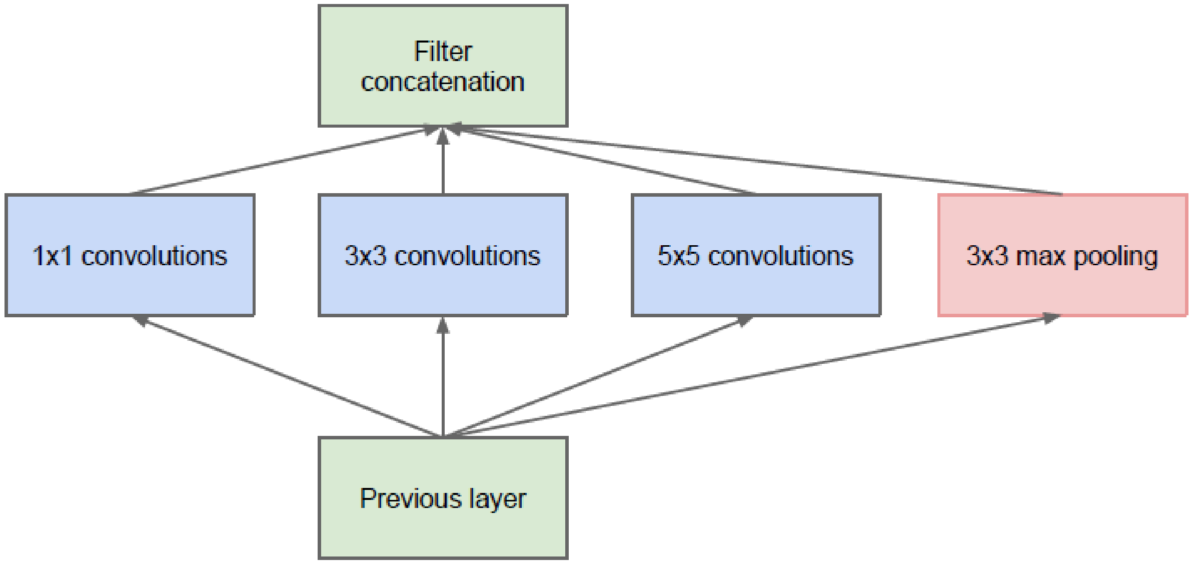


- **Inception V3 Pretrained on Image Net**



We conclude that running data on pretrained model work very well than not pretrained model,

And using The Inception V3 model used several techniques for optimizing the network for better model adaptation. It has a deeper network compared to the Inception V1 and V2 models, but its speed isn't compromised. It is computationally less expensive. It uses auxiliary Classifiers as regularizes.



Snapshot from IncpetionV3 Code:

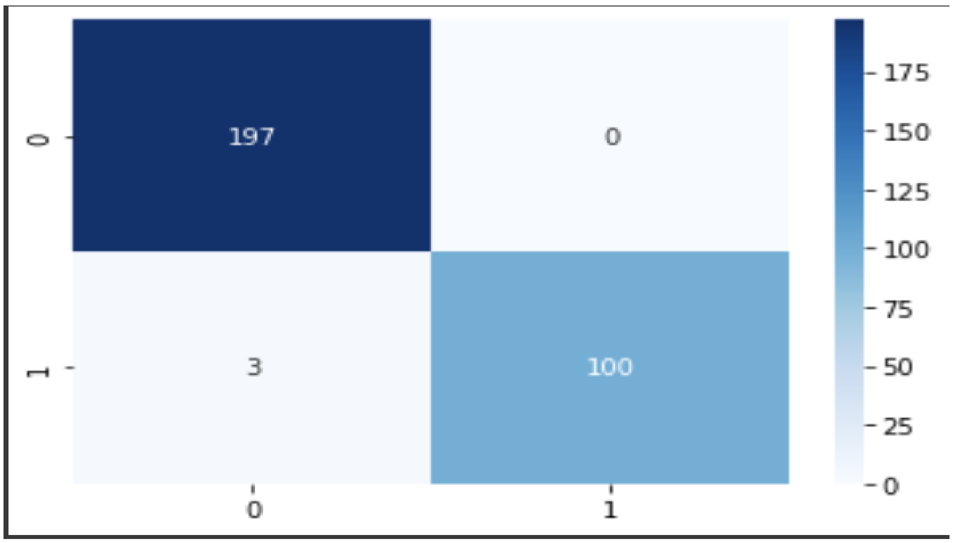
```
# create the base pre-trained model
base_model = InceptionV3(weights='imagenet', include_top=False)

# add a global spatial average pooling layer

x = base_model.output
x = GlobalAveragePooling2D()(x)
# let's add a fully-connected layer
x = Dense(1024, activation='relu')(x)


predictions = Dense(1, activation='sigmoid')(x)

model = Model(inputs=base_model.input, outputs=predictions)

# i.e. freeze all convolutional InceptionV3 layers
for layer in base_model.layers:
    layer.trainable = False

# compile the model (should be done *after* setting layers to non-trainable)
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy',metrics=['accuracy',tf.keras.metrics.Recall(),tf.keras.metrics.Precision()])

# train the model on the new data for a few epochs
model.fit(X, Y1, batch_size=32, epochs=15, validation_split=0.2,verbose=1)
score = model.evaluate(x_test, Ys1_test, verbose=1)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}, Recall: {score[2]}, Precision: {score[3]}')
```

- Results

  As shown the Resnet50 preform better than the Inception

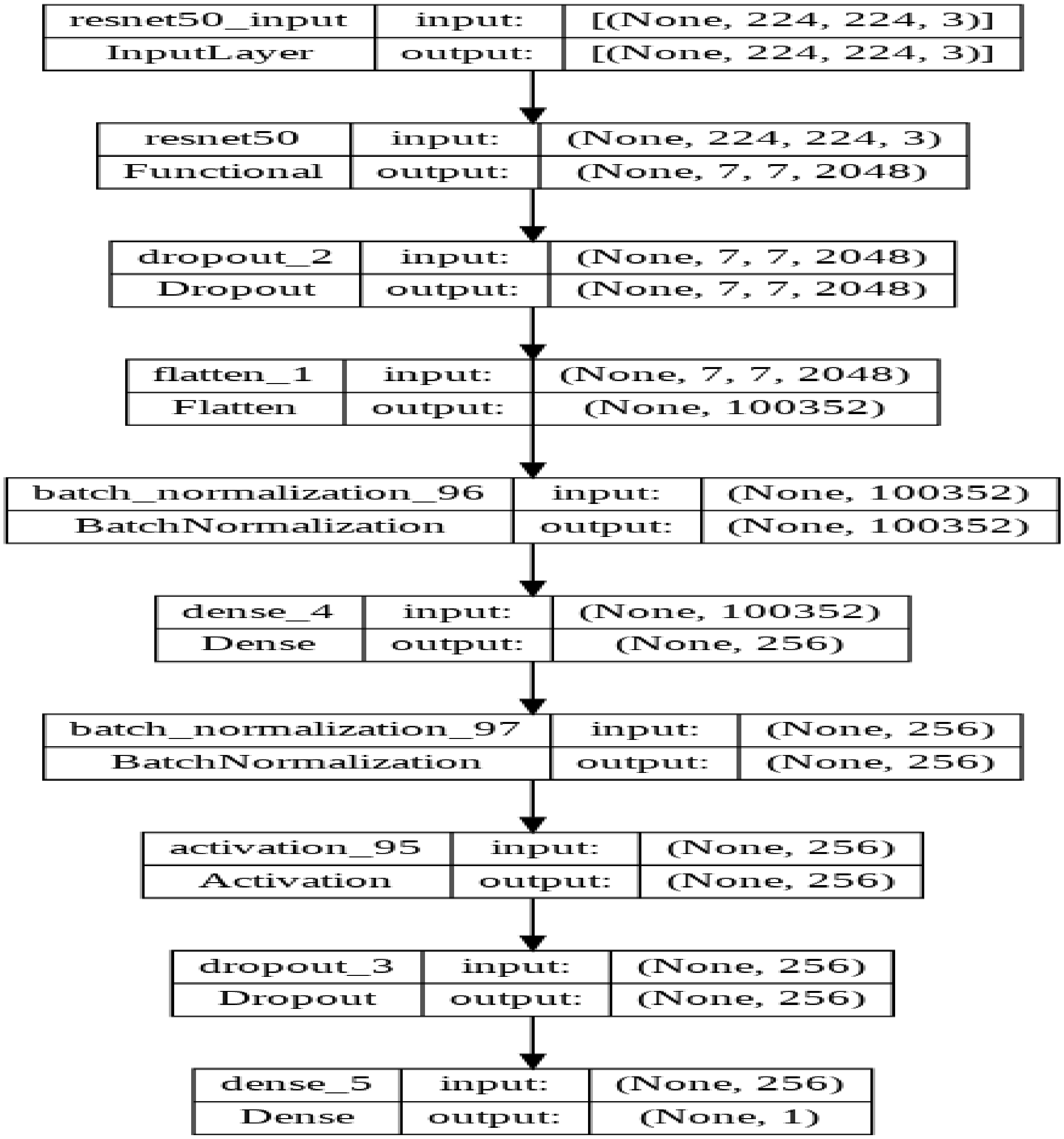
```
Test loss: 0.08544208854436874
Test accuracy: 0.9900000095367432
Recall: 1.0
Precision: 0.9708737730979919
```

As shown in confusion matrix there is brain images predicted as Breast which leads to decrease in precsion.



- Second level classification:
    - Brain (Tumor or No-Tumor):
        - Resnet 50 Pretrained on Image net

          The 50-layer ResNet uses a bottleneck design for the building block. A bottleneck residual block



Results:

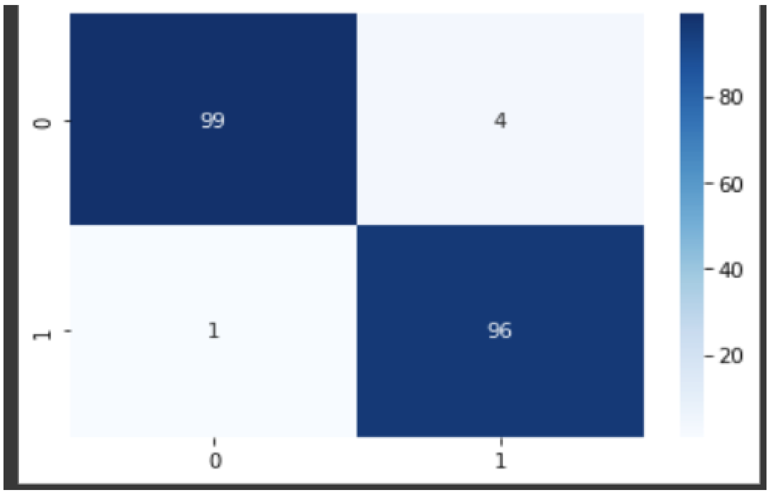
As shown in table the results of train and validation is good there is only little misclassification.

Minimum training loss	Minimum Val. Loss	Maximum training accuracy	Maximum val. accuracy	Max training recall	Max val. recall
0.001702	0.030827	1.000000	0.985075	1.000000	0.989899

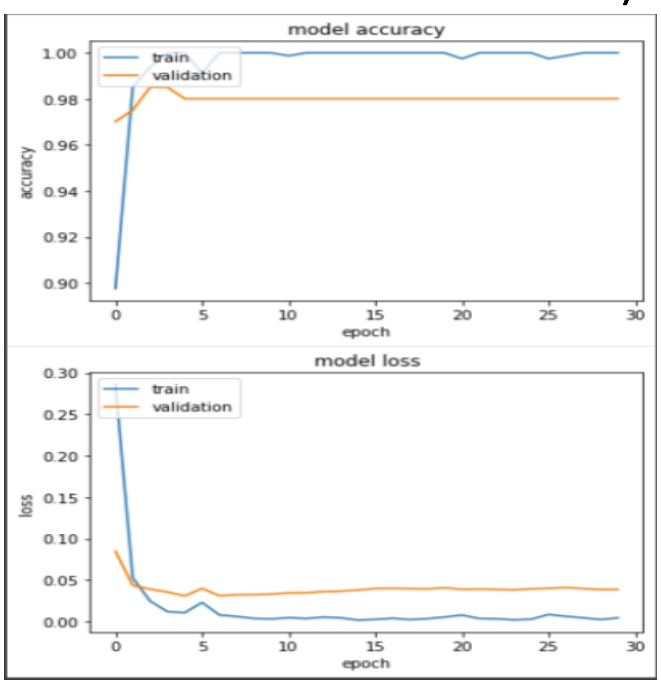
The test there is a little bit of misclassification, and it can be more clear in the confusion matrix

Test loss	Test accuracy	Recall	Precision
0.085742	0.975000	0.960000	0.989691

Confusion Matrix



Train vs Validation in Accuracy and Loss



- Breast (Normal, Benign and Malignant).
    - Mobile NetV1 Pretrained on Image net +Up Sampling(Best Model)

We used mobile net as it fewer network, don't contain a lot of parameters, contain residual network and it gives higher classification accuracy.

```
[ ]
from keras.applications import MobileNet
base_model=MobileNet(input_shape=(IMG_SIZE,IMG_SIZE,3),weights='imagenet',include_top=False) #imports the mobilenet model and discards the last 1000 neuron layer.

x=base_model.output
x=GlobalAveragePooling2D()(x)
x=Dense(1024,activation='relu')(x) #we add dense layers so that the model can learn more complex functions and classify for better results.
x=Dense(1024,activation='relu')(x) #dense layer 2
x=Dense(512,activation='relu')(x) #dense layer 3
preds=Dense(3,activation='softmax')(x) #final layer with softmax activation

model=Model(inputs=base_model.input,outputs=preds)

model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy' , tf.keras.metrics.Recall(),tf.keras.metrics.Precision()])
# Adam optimizer
# loss function will be categorical cross entropy
# evaluation metric will be accuracy
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.3,
                              patience=3, min_lr=0.0000001)
model.fit(X_Breast_Train, Y_Breast_Train,epochs=40,validation_split=0.05,verbose=1,callbacks=[reduce_lr])
score = model.evaluate(X_Breast_Test, Y_Breast_Test, verbose=1)


print(f'Test loss: {score[0]} / Test accuracy: {score[1]}, Recall: {score[2]}, Precision: {score[3]}')
```

Results:

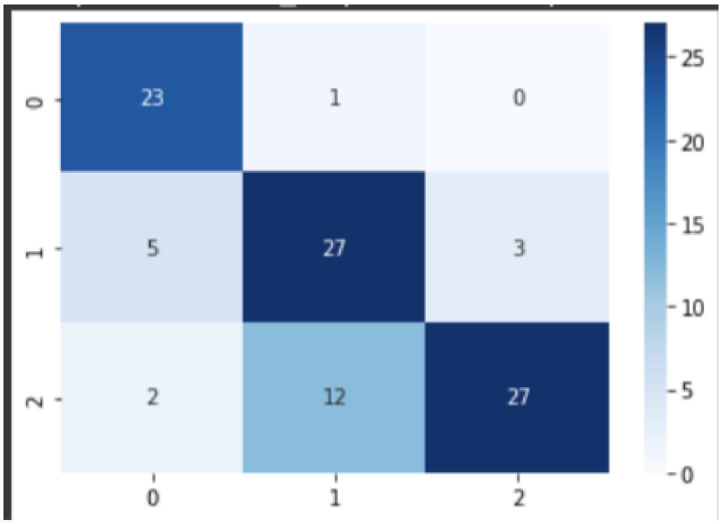
Train and Validation Results:

Minimum training loss	Minimum Val. Loss	Maximum training accuracy	Maximum val. accuracy	Max training recall	Max val. recall
0.000597	0.287011	1.000000	0.966667	1.000000	0.966667

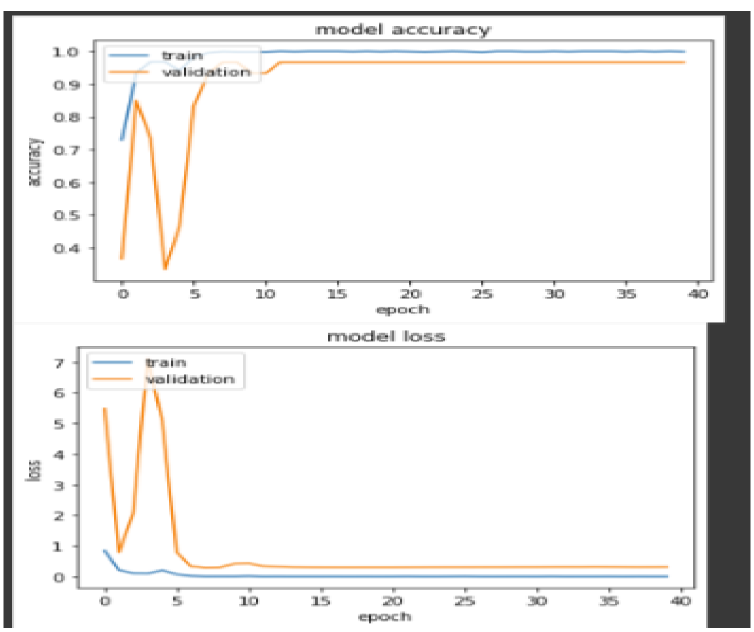
Test Results:

Test loss	Test accuracy	Recall	Precision
1.323240	0.770000	0.770000	0.777778

Confusion Matrix:

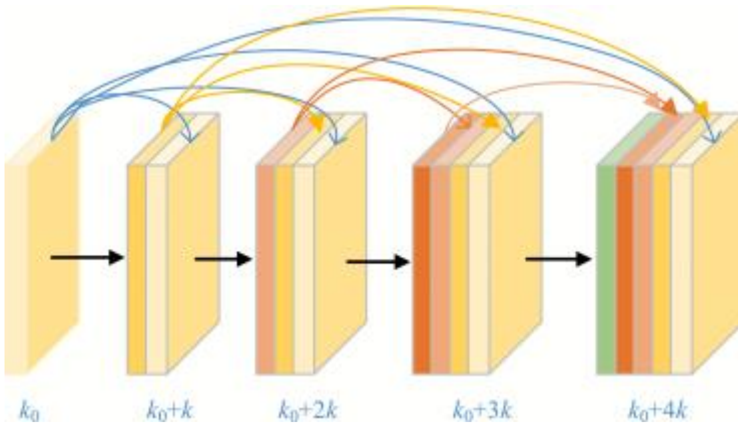


Train Vs Validation in accuracy and loss



- Dense Net 201(Second Best One)

Dense Nets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters.

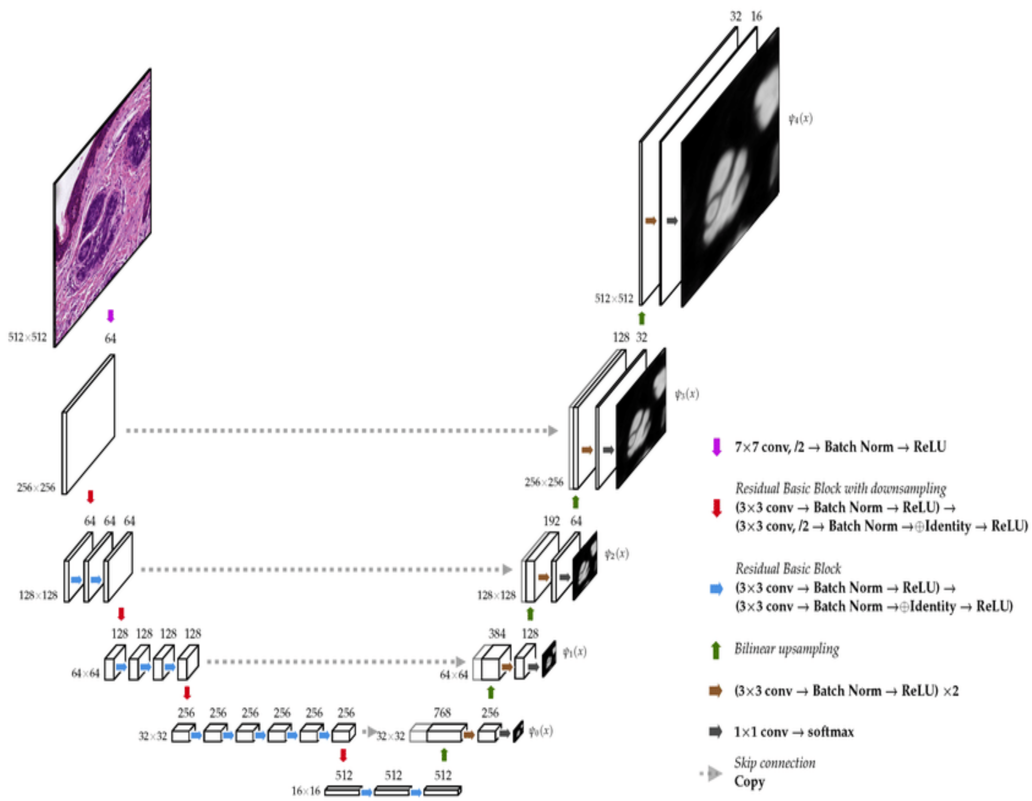


Snapshot from the DenseNet201 code:

```
## evaluate

pretrained_densenet = tf.keras.applications.DenseNet201(input_shape=(IMG_SIZE, IMG_SIZE, 3), weights='imagenet', include_top=False)

for layer in pretrained_densenet.layers:
  layer.trainable = True

x1 = pretrained_densenet.output
x1 = tf.keras.layers.Dropout(0.2, name="dropout_head_1")(x1)
x1 = tf.keras.layers.AveragePooling2D(name="averagepooling2d_head")(x1)
x1 = tf.keras.layers.Flatten(name="flatten_head")(x1)
x1 = tf.keras.layers.Dense(64, activation="relu", name="dense_head")(x1)
x1 = tf.keras.layers.Dropout(0.2, name="dropout_head_2")(x1)
model_out = tf.keras.layers.Dense(3, activation='softmax', name="predictions_head")(x1)

model_densenet = Model(inputs=pretrained_densenet.input, outputs=model_out)
model_densenet.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy' , tf.keras.metrics.Recall(),tf.keras.metrics.Precision()])

reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.3,
                              patience=3, min_lr=0.0000001)

model_densenet.fit(X_Breast_Train, Y_Breast_Train,epochs=80,validation_split=0.05,verbose=1,callbacks=[reduce_lr])
score = model_densenet.evaluate(X_Breast_Test, Y_Breast_Test, verbose=1)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}, Recall: {score[2]}, Precision: {score[3]}')
```

- Segmentation level:

- Brain mask segmentation.
  - **UNET +ResNet34**
    - We used Segmentation model Library to get pretrained model
  - We used “ResNet34” as a backbone pretrained network to make feature-extraction and processes input data certain feature representation, after the pre-trained model we used “UNet” model and used Weights of “ImageNet”.



Snapshot from the pretrained UNet code:

```
#!pip install segmentation_models
import segmentation_models as sm

BACKBONE = 'resnet34'
preprocess_input = sm.get_preprocessing(BACKBONE)

# load your data

# preprocess input
x_train = preprocess_input(x_brain_seg)
#x_val = preprocess_input(x_val)

# define model
model = sm.Unet(BACKBONE, encoder_weights='imagenet',classes=1, activation='sigmoid')
model.compile(
    'Adam',
    loss="binary_crossentropy",
    metrics=[iou_coef],
)


tf.cast(x_brain_seg, tf.float32)
train_brain_mask=tf.cast(train_brain_mask, tf.float32)


model.fit(x_brain_seg,train_brain_mask,validation_split=0.1,

   batch_size=16,
   epochs=100,
```

Results:

Train and Validation Results:

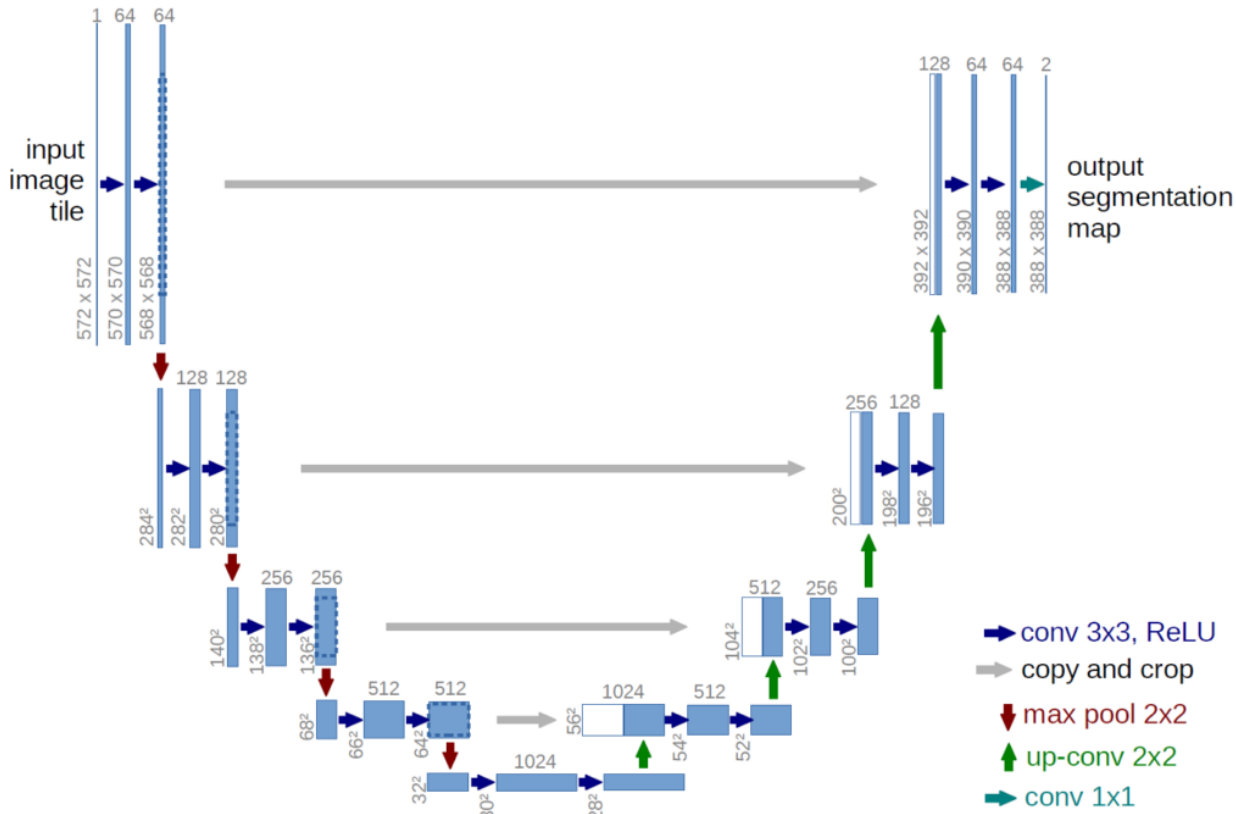
	Max. training loss	Minimum Val. Loss	Maximum training IoU	Maximum val. accuracy	Max Dice coef	Max Dice coef
0	0.391102	9.746922	0.967586	0.862126	0.953584	0.790039

Test Validation:

Test loss	IoU	Dice Coef
0.062130	0.844130	0.817119

- **Breast mask segmentation**.
  - **UNET+MobileNet**
    - We used Segmentation model Library to get pretrained model

- We used “Mobile Net” as a backbone pretrained network to make feature-extraction and processes input data into a certain feature representation, after the pre-trained model we used “UNet” model and used Weights of “ImageNet”.



- **UNET+ efficientnetb2 (Best One)**
  - We used Segmentation model Library to get pretrained model

- We used “efficientnetb2” as a backbone pretrained network to make feature-extraction and processes input data into a certain feature representation, after the pre-trained model we used “UNet” model and used Weights of “ImageNet”.

Results:

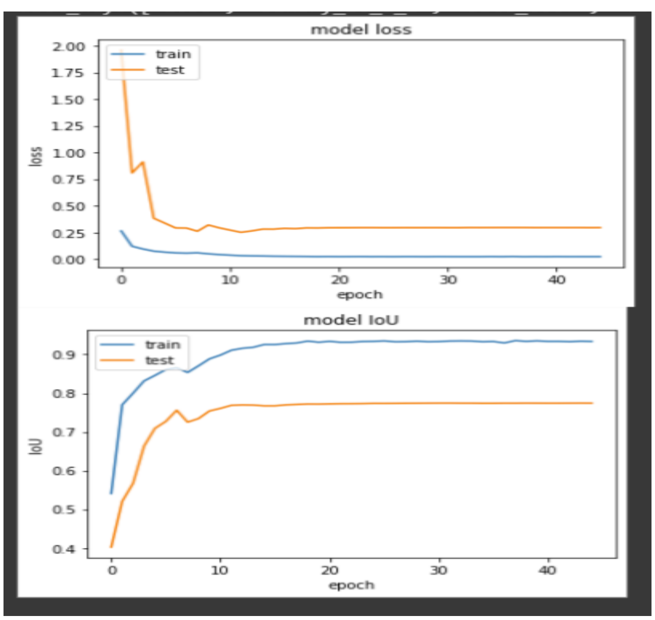
Train and Validation results

	Max. training loss	Minimum Val. Loss	Maximum training IoU	Maximum val. accuracy	Max Dice coef	Max Dice coef
0	0.264377	1.958497	0.935683	0.77429	0.902363	0.718776

Test Results:

Test loss	IoU	Dice Coef
0.251991	0.751567	0.748941

Train vs Validation Accuracy and loss:



- Training and Testing Time (Using GoogleColab GPU)

	Training time	**Testing time**
Brain-Breast Classification	0:03:26	00:00:3
Brain classification **(Tumor-No tumor)**	0:02:25	00:00:02
Breast classification **(Normal, Benign or Malignant)**	0:05:25	00:00:01
Brain Segmentation	0:06:34	00:00:03
Breast Segmentation	0:09:36	00:00:05