In [1]:

```
import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')
```

In [2]:

```
import os
import numpy as np
np.random.seed(777)
import math
import keras
import keras.backend as K
import h5py
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from keras.models import Sequential
from keras.models import Model
from keras.layers import Input, Activation, merge, Dense, Flatten, Dropout, concatenate
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.layers import BatchNormalization, add, GlobalAveragePooling2D
from keras.utils.np_utils import to_categorical
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import accuracy_score,roc_curve, confusion_matrix, roc_auc_score, auc,
from keras.regularizers import l2
from keras.applications.xception import Xception, preprocess_input

from keras.layers import Input, Dense, Activation, ZeroPadding2D, BatchNormalization, Flatt
from keras.layers import SeparableConv2D, AveragePooling2D, MaxPooling2D, Dropout, GlobalMa

import matplotlib.pyplot as plt
%matplotlib inline

plt.rcParams["axes.grid"] = False
plt.rcParams.update({'font.size': 20})
```

```
INFO:tensorflow:Enabling eager execution
INFO:tensorflow:Enabling v2 tensorshape
INFO:tensorflow:Enabling resource variables
INFO:tensorflow:Enabling tensor equality
INFO:tensorflow:Enabling control flow v2
```

In [3]:

```
train_dir = 'C://Users//Mrinal Anand//Desktop//Dataset_tumor//train'
test_dir = 'C://Users//Mrinal Anand//Desktop//Dataset_tumor//test'

extracted_features_dir = 'C://Users//Mrinal Anand//Desktop//extracted_features//'
model_name = "Xception_concate"
```

In [4]:

```
import keras
import tensorflow as tf
import keras.backend as K

print("Keras Version", keras.__version__)
print("tensorflow Version", tf.__version__)
print("dim_ordering:", K.image_data_format)
```

```
Keras Version 2.5.0
tensorflow Version 2.5.0-rc0
dim_ordering: <function image_data_format at 0x0000020C44D7FC10>
```

In [5]:

```
batch_size = 32
img_height, img_width = 224, 224
input_shape = (img_height, img_width, 3)
epochs = 1000
```

In [6]:

```
for root,dirs,files in os.walk(train_dir):
    print (root, len(files))



print("*"*30)
for root,dirs,files in os.walk(test_dir):
    print (root, len(files))
```

```
C://Users//pauls//Desktop//Dataset_tumor//Train 0
C://Users//pauls//Desktop//Dataset_tumor//Train\benign 6128
C://Users//pauls//Desktop//Dataset_tumor//Train\malignant 2976
******************************
C://Users//pauls//Desktop//Dataset_tumor//Test 0
C://Users//pauls//Desktop//Dataset_tumor//Test\benign 126
C://Users//pauls//Desktop//Dataset_tumor//Test\malignant 51
```

In [7]:

```
random_seed = np.random.seed(1142)

train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    featurewise_center=True,
    featurewise_std_normalization=True,
    validation_split= 0.25,
    zoom_range=0.2,
    shear_range=0.2)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    seed = random_seed,
    shuffle = False,
    subset = 'training',
    class_mode='categorical')

validation_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    seed = random_seed,
    shuffle = False,
    subset = 'validation',
    class_mode='categorical')

test_datagen = ImageDataGenerator(rescale=1. / 255)
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    seed = random_seed,
    shuffle = False,
    class_mode='categorical')
```

```
Found 6828 images belonging to 2 classes.
Found 2276 images belonging to 2 classes.
Found 177 images belonging to 2 classes.
```

In [8]:

```
nb_train_samples = len(train_generator.filenames)
nb_validation_samples = len(validation_generator.filenames)
predict_size_train = int(math.ceil(nb_train_samples / batch_size))
predict_size_validation = int(math.ceil(nb_validation_samples / batch_size))

nb_test_samples = len(test_generator.filenames)
predict_size_test = int(math.ceil(nb_test_samples / batch_size))



num_classes = len(train_generator.class_indices)

print("nb_train_samples:", nb_train_samples)
print("nb_validation_samples:", nb_validation_samples)
print("\npredict_size_train:", predict_size_train)
print("predict_size_validation:", predict_size_validation)

print("nb_test_samples:", nb_test_samples)
print("predict_size_test:", predict_size_test)

print("\n num_classes:", num_classes)
```

```
nb_train_samples: 6828
nb_validation_samples: 2276

predict_size_train: 214
predict_size_validation: 72
nb_test_samples: 177
predict_size_test: 6

 num_classes: 2
```

In [9]:

```
from keras.backend import get_session
from keras.backend import clear_session
from keras.backend import set_session

def reset_keras_tf_session():
    """
    this function clears the gpu memory and set the
    tf session to not use the whole gpu
    """
    sess = get_session()
    clear_session()
    sess.close()
    sess = get_session()

    config = tf.compat.v1.ConfigProto()
    config.gpu_options.allow_growth = True
    set_session(tf.compat.v1.Session(config=config))


reset_keras_tf_session()
```

In [10]:

```
model = Xception(weights='imagenet', include_top=False, pooling = 'avg',input_tensor=Input(
```

In [11]:

```
for i, layer in enumerate(model.layers):
    print(i, layer.name)
```

```
0 input_1
1 block1_conv1
2 block1_conv1_bn
3 block1_conv1_act
4 block1_conv2
5 block1_conv2_bn
6 block1_conv2_act
7 block2_sepconv1
8 block2_sepconv1_bn
9 block2_sepconv2_act
10 block2_sepconv2
11 block2_sepconv2_bn
12 conv2d
13 block2_pool
14 batch_normalization
15 add
16 block3_sepconv1_act
17 block3_sepconv1
18 block3_sepconv1_bn
19 block3_sepconv2_act
```

In [12]:

```
c1 = model.layers[16].output
c1 = GlobalAveragePooling2D()(c1)

c2 = model.layers[26].output
c2 = GlobalAveragePooling2D()(c2)

c3 = model.layers[36].output
c3 = GlobalAveragePooling2D()(c3)

c4 = model.layers[126].output
c4 = GlobalAveragePooling2D()(c4)

con = concatenate([c2, c3, c4])

bottleneck_final_model = Model(inputs=model.input, outputs=con)
```

In [13]:

```
bottleneck_final_model.summary()
```

```
Model: "model"
__________________________________________________________________________
________________________
Layer (type)                    Output Shape         Param #     Connecte
d to
==========================================================================
========================
input_1 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________
________________________
block1_conv1 (Conv2D)           (None, 111, 111, 32) 864         input_1
[0][0]
__________________________________________________________________________
________________________
block1_conv1_bn (BatchNormaliza (None, 111, 111, 32) 128         block1_c
onv1[0][0]
__________________________________________________________________________
________________________
block1_conv1_act (Activation)   (None, 111, 111, 32) 0           block1_c
```

In [14]:

```
bottleneck_features_train = bottleneck_final_model.predict_generator(train_generator, predi
np.save(extracted_features_dir+'bottleneck_features_train_'+model_name+'.npy', bottleneck_f
```

In [15]:

```
bottleneck_features_validation = bottleneck_final_model.predict_generator(validation_genera
np.save(extracted_features_dir+'bottleneck_features_validation_'+model_name+'.npy', bottlen

bottleneck_features_test = bottleneck_final_model.predict_generator(test_generator, predict
np.save(extracted_features_dir+'bottleneck_features_test_'+model_name+'.npy', bottleneck_fe
```

In [16]:

```
train_data = np.load(extracted_features_dir+'bottleneck_features_train_'+model_name+'.npy')
validation_data = np.load(extracted_features_dir+'bottleneck_features_validation_'+model_na
test_data = np.load(extracted_features_dir+'bottleneck_features_test_'+model_name+'.npy')

train_labels = train_generator.classes
train_labels = to_categorical(train_labels, num_classes=num_classes)

validation_labels = validation_generator.classes
validation_labels = to_categorical(validation_labels, num_classes=num_classes)

test_labels = test_generator.classes
test_labels = to_categorical(test_labels, num_classes=num_classes)
```

In [17]:

```
dropout_rate = 0.5

model = Sequential()
model.add(Dense(256, activation='relu'))
model.add(Dropout(dropout_rate))
model.add(Dense(num_classes, activation=tf.nn.softmax))

adam = Adam(lr = 0.001, beta_1=0.6, beta_2=0.8, amsgrad=True)

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

history = model.fit(train_data, train_labels,
                    epochs=epochs,
                    batch_size=batch_size,
                    validation_data=(validation_data, validation_labels),
                    verbose= 2)

with open(extracted_features_dir+'history_'+model_name+'.txt','w') as f:
    f.write(str(history.history))
```

```
Epoch 1/1000
WARNING:tensorflow:AutoGraph could not transform <bound method Dense.call
of <keras.layers.core.Dense object at 0x0000020C48229A90>> and will run i
t as-is.
Please report this to the TensorFlow team. When filing the bug, set the v
erbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the
full output.
Cause: invalid syntax (tmpca8p7n5k.py, line 48)
To silence this warning, decorate the function with @tf.autograph.experim
ental.do_not_convert
WARNING: AutoGraph could not transform <bound method Dense.call of <kera
s.layers.core.Dense object at 0x0000020C48229A90>> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the v
erbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the
full output.
Cause: invalid syntax (tmpca8p7n5k.py, line 48)
To silence this warning, decorate the function with @tf.autograph.experim
ental.do_not_convert
214/214 - 19s - loss: 0.4881 - accuracy: 0.7907 - val_loss: 0.3689 - val_
          0.8445
```

In [18]:

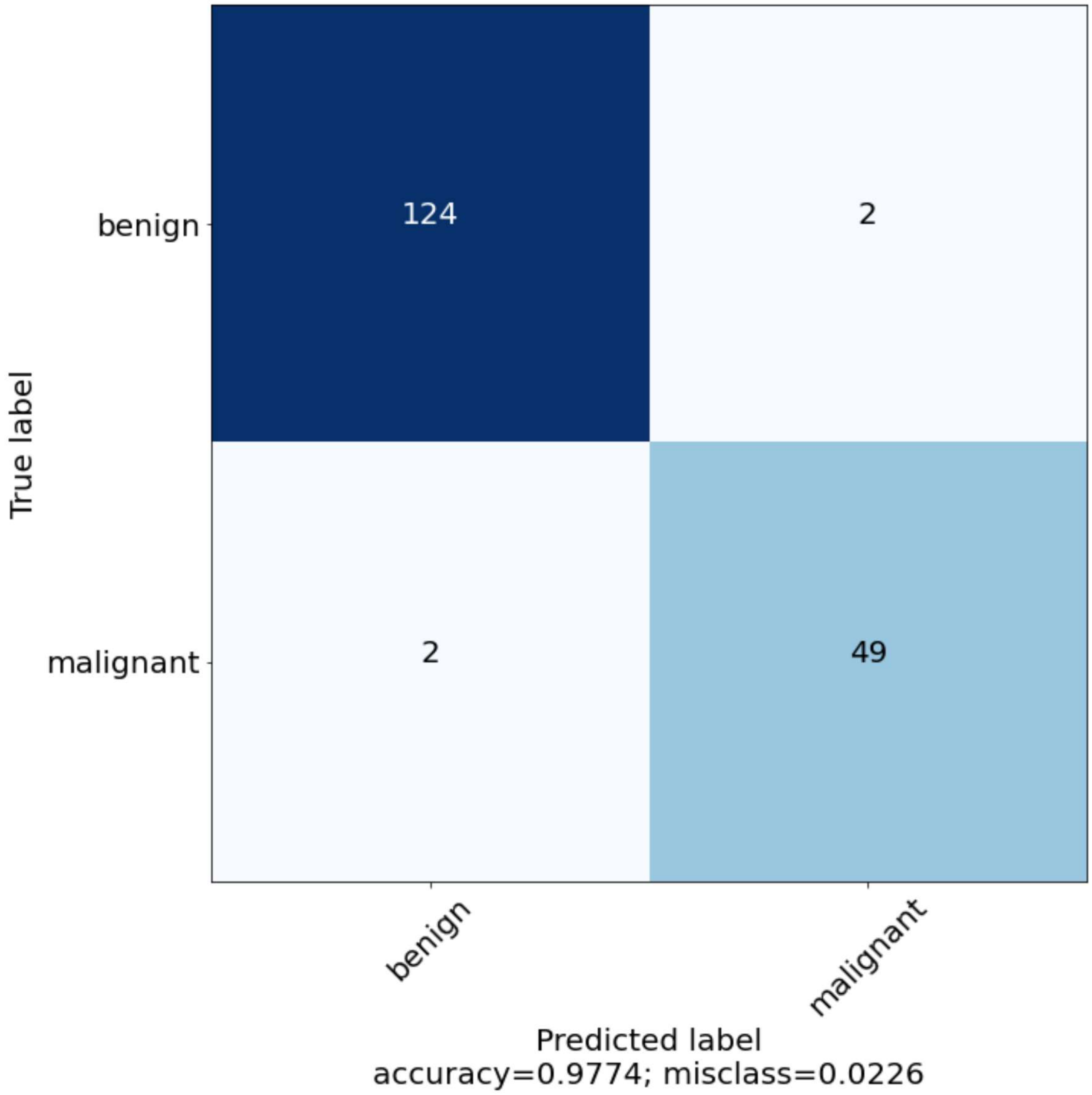
```
preds = model.predict(test_data)

predictions = [i.argmax() for i in preds]
y_true = [i.argmax() for i in test_labels]
cm = confusion_matrix(y_pred=predictions, y_true=y_true)

print('Accuracy {}'.format(accuracy_score(y_true=y_true, y_pred=predictions)))
```

```
Accuracy 0.9774011299435028
```

In [19]:

```
plt.rcParams["axes.grid"] = False
plt.rcParams.update({'font.size': 20})

labels = []

label = test_generator.class_indices
indexlabel = dict((value, key) for key, value in label.items())

for k,v in indexlabel.items():
    labels.append(v)

from sklearn.metrics import confusion_matrix
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion Matrix')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(accuracy, miscl



plt.figure(figsize=(10,10))
plot_confusion_matrix(cm, classes=labels, title=' ')
```

```
Confusion Matrix
[[124   2]
 [  2  49]]
```


True label
benign
malignant
124
2
2
49
benign
malignant
Predicted label
accuracy=0.9774; misclass=0.0226

In [20]:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report
y_pred=predictions
y_pred_probabilities=y_pred
y_actual = y_true

classnames=[]
for classname in test_generator.class_indices:
    classnames.append(classname)

confusion_mtx = confusion_matrix(y_actual, y_pred)
print(confusion_mtx)
target_names = classnames
print(classification_report(y_actual, y_pred, target_names=target_names))
```

```
[[124   2]
 [  2  49]]
              precision    recall  f1-score   support

      benign       0.98      0.98      0.98       126
   malignant       0.96      0.96      0.96        51

    accuracy                           0.98       177
   macro avg       0.97      0.97      0.97       177
weighted avg       0.98      0.98      0.98       177
```

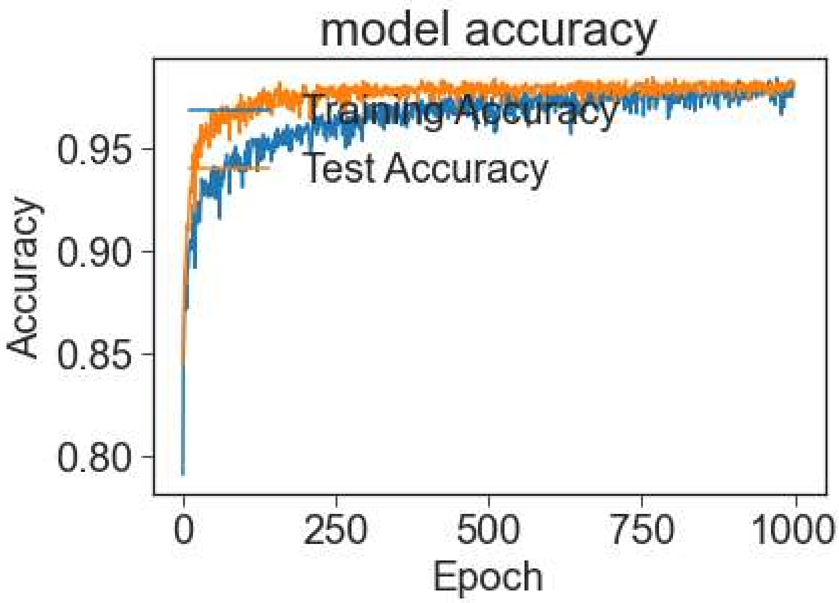
In [21]:

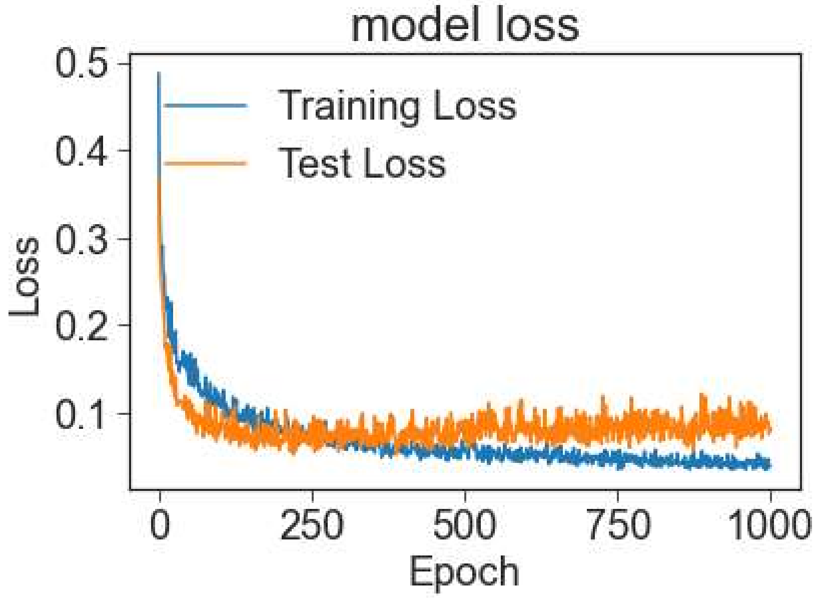
```
total=sum(sum(cm))

sensitivity = cm[0,0]/(cm[0,0]+cm[1,0])
print('Sensitivity : ', sensitivity*100 )

Specificity = cm[1,1]/(cm[1,1]+cm[0,1])
print('Specificity : ', Specificity*100 )
```

```
Sensitivity :  98.4126984126984
Specificity :  96.07843137254902
```

In [22]:

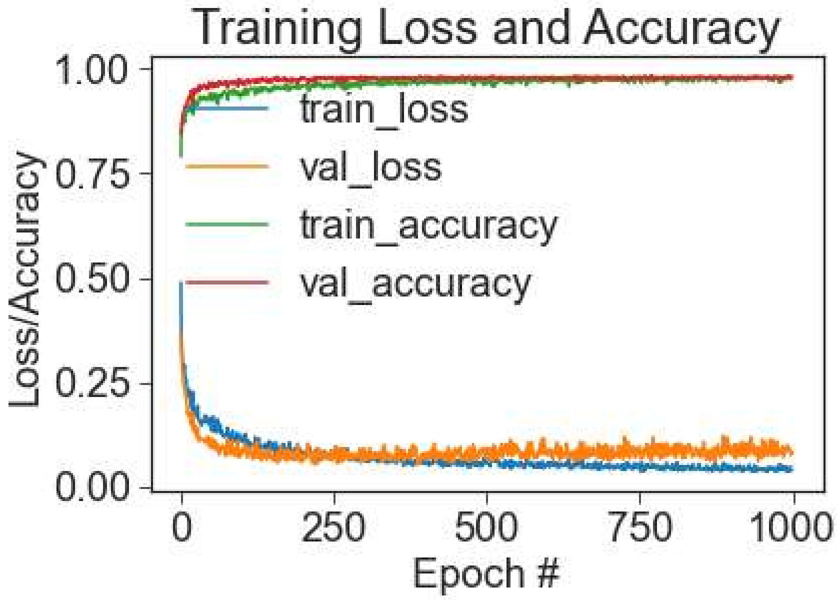
```
plt.style.use("seaborn-ticks")

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training Accuracy', 'Test Accuracy'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training Loss', 'Test Loss'], loc='upper left')
plt.show()

plt.figure()
N = epochs
plt.plot(np.arange(0, N), history.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), history.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), history.history["accuracy"], label="train_accuracy")
plt.plot(np.arange(0, N), history.history["val_accuracy"], label="val_accuracy")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="upper left")
```





Out[22]:

```
<matplotlib.legend.Legend at 0x20c0152f970>
```



In [ ]:

In [ ]:

In [ ]:

In [ ]: