In [1]:

```
from google.colab import drive

drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [2]:

```
%cd /content/gdrive/My Drive/Cornell/2020-2021/Biomedical ML Final Project - Melted Paper/
```

```
/content/gdrive/.shortcut-targets-by-id/13OShW0589KdYJRWaN9GrhtF2vtwdrUUW/Biomedical ML Final
Project - Melted Paper
```

Augment Dataset

Perform random number of image transformations to balance the classes

In [3]:

```
import os, time, cv2, random
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
import skimage as sk
import skimage.transform
```

In [4]:

```
augment_path = os.path.join('Datasets', 'final-dataset-augment')
normal_images_path = os.path.join('Datasets', 'final-dataset', 'N')
benign_images_path = os.path.join('Datasets', 'final-dataset', 'B')
malignant_images_path = os.path.join('Datasets', 'final-dataset', 'M')
```

In [5]:

```
normal_images = os.listdir(normal_images_path)
benign_images = os.listdir(benign_images_path)
malignant_images = os.listdir(malignant_images_path)
```

In [6]:

```
images, labels = [], []
```

In [7]:

```
for img in normal_images:
  labels.append('N')
  image = cv2.imread(os.path.join(normal_images_path, img), cv2.IMREAD_GRAYSCALE)
  image = image.reshape((image.shape[0], image.shape[1], 1))
  images.append(image)
```

In [8]:

```
for img in benign_images:
  labels.append('B')
  image = cv2.imread(os.path.join(benign_images_path, img), cv2.IMREAD_GRAYSCALE)
  image = image.reshape((image.shape[0], image.shape[1], 1))
  images.append(image)
```

In [9]:

```
for img in malignant_images:
  labels.append('M')
  image = cv2.imread(os.path.join(malignant_images_path, img), cv2.IMREAD_GRAYSCALE)
  image = image.reshape((image.shape[0], image.shape[1], 1))
  images.append(image)
```

In [10]:

```
le = LabelEncoder()
le_labels = to_categorical(le.fit_transform(labels))
```

In [11]:

```
def get_class_count(labels):
  '''
  Get the number of images in each class.
  '''
  num_classes = len(labels[0])
  counts = np.zeros(num_classes)
  for label in labels:
    for i in range(num_classes):
      counts[i] += label[i]
  return counts.tolist()
```

In [12]:

```
def random_shearing(img):
  tf = sk.transform.AffineTransform(shear=random.uniform(-0.3, 0.3))
  return sk.transform.warp(img, tf, order=1, preserve_range=True, mode='wrap')

def random_noise(img):
  return sk.util.random_noise(img)

def random_rotation(img):
  return sk.transform.rotate(img, random.uniform(-30, 30))

def horizontal_flip(img):
  return img[:, ::-1]
```

In [13]:

```
transformation_functions = {
  'shear': random_shearing,
  'rotate': random_rotation,
  'noise': random_noise,
  'horizontal_flip': horizontal_flip,
}
```

In [14]:

```
def transform_images(img, transforms: dict):
  '''
  Perform a random number of image transformations.
  '''
  num_transformations = random.randint(0, len(transforms))
  transformed_image = img
  for i in range(num_transformations):
    key = random.choice(list(transforms))
    transformed_image = transforms[key](img)

  return transformed_image
```

In [15]:

```
def generate_more_images(images, labels, transformation_functions):
  '''
  Determine the number of images needed in each class for balance,
  then transform images to augment.
  '''
```

```
  more_images = images
  more_labels = labels

  class_balance = get_class_count(labels)
  img_to_add = [max(class_balance) - i for i in class_balance]

  for i in range(len(img_to_add)):
    if int(img_to_add[i]) == 0:
      continue
    label = np.zeros(len(img_to_add))
    label[i] = 1
    class_label_indices = [i for i, x in enumerate(labels) if np.array_equal(x, label)]
    class_images = [images[i] for i in class_label_indices]

    for k in range(int(img_to_add[i])):
      transformed_image = transform_images(class_images[k % len(class_images)],
transformation_functions)
      transformed_image = transformed_image.reshape(1, transformed_image.shape[0], transformed_imag
e.shape[1], 1)

      more_images = np.append(more_images, transformed_image, axis=0)
      more_labels = np.append(more_labels, label.reshape(1, len(label)), axis=0)
  return more_images, more_labels
```

In [16]:

```
augmented_images, augmented_labels = generate_more_images(images, le_labels,
transformation_functions)
```

In [17]:

```
print('Original number of images: {}'.format(len(augmented_images)))
print('Augmented number of images: {}'.format(len(images)))
```

```
Original number of images: 3948
Augmented number of images: 2616
```

In [ ]:

```
for i in range(len(augmented_images)):
  path = os.path.join(augment_path, augmented_labels[i], 'image' + str(i) + '.jpg')
  cv2.imwrite(path, augmented_images[i])
```

In [4]:

```
from google.colab import drive

drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [5]:

```
%cd /content/gdrive/My Drive/Cornell/2020-2021/Biomedical ML Final Project - Melted Paper/
```

```
/content/gdrive/.shortcut-targets-by-id/13OShW0589KdYJRWaN9GrhtF2vtwdrUUW/Biomedical ML Final
Project - Melted Paper
```

Compile Dataset

Combine images from MIAS, INbreast, DDSM

In [6]:

```
import os, time, cv2, random
import numpy as np
import pandas as pd
```

In [7]:

```
all_mias_path = os.path.join('Datasets', 'archive', 'sam', 'all-mias')
ddsm_path = os.path.join('Datasets', 'archive', 'full-dataset', 'DDSM Dataset')
inbreast_path = os.path.join('Datasets', 'archive', 'full-dataset', 'INbreast Dataset')
final_path = os.path.join('Datasets', 'final-dataset')
```

In [9]:

```
df = pd.read_table(os.path.join(all_mias_path, 'Info.txt'), delimiter=' ')
df.SEVERITY = df.SEVERITY.fillna('N')
lookUp = df[['REFNUM', 'SEVERITY']].set_index('REFNUM').T.to_dict()
df.head()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: UserWarning: DataFrame columns are
not unique, some columns will be omitted.
  This is separate from the ipykernel package so we can avoid doing imports until
```

Out[9]:

	REFNUM	BG	CLASS	SEVERITY	X	Y	RADIUS	Unnamed: 7
0	mdb001	G	CIRC	B	535.0	425.0	197.0	NaN
1	mdb002	G	CIRC	B	522.0	280.0	69.0	NaN
2	mdb003	D	NORM	N	NaN	NaN	NaN	NaN
3	mdb004	D	NORM	N	NaN	NaN	NaN	NaN
4	mdb005	F	CIRC	B	477.0	133.0	30.0	NaN

In [10]:

```
all_mias = [img for img in os.listdir(all_mias_path) if img.endswith('.jpg')]
ddsm_benign = [img for img in os.listdir(os.path.join(ddsm_path, 'Benign Masses')) if img.endswith(
'.png') and ' ' not in img]
ddsm_malignant = [img for img in os.listdir(os.path.join(ddsm_path, 'Malignant Masses')) if img.end
swith('.png') and ' ' not in img]
inbreast_benign = [img for img in os.listdir(os.path.join(inbreast_path, 'Benign Masses')) if img.e
ndswith('.png') and ' ' not in img]
```

```
ndswith('.png') and ' ' not in img]
inbreast_malignant = [img for img in os.listdir(os.path.join(inbreast_path, 'Malignant Masses')) if
img.endswith('.png') and ' ' not in img]
```

In [14]:

```
print('MIAS Dataset \n --------------')
print('Number of normal images: {}'.format(sum(x['SEVERITY'] == 'N' for x in lookUp.values())))
print('Number of benign images: {}'.format(sum(x['SEVERITY'] == 'B' for x in lookUp.values())))
print('Number of malignant images: {}'.format(sum(x['SEVERITY'] == 'M' for x in lookUp.values())))
print('\nINbreast Dataset \n --------------')
print('Number of benign images: {}'.format(len(inbreast_benign)))
print('Number of malignant images: {}'.format(len(inbreast_malignant)))
print('\nDDSM Dataset \n --------------')
print('Number of benign images: {}'.format(len(ddsm_benign)))
print('Number of malignant images: {}'.format(len(ddsm_malignant)))
```

```
MIAS Dataset
 --------------
Number of normal images: 207
Number of benign images: 63
Number of malignant images: 52

INbreast Dataset
 --------------
Number of benign images: 35
Number of malignant images: 71

DDSM Dataset
 --------------
Number of benign images: 995
Number of malignant images: 1193
```

In [15]:

```
target_size = (224, 224)
```

In [16]:

```
for img in all_mias:
  img_original = cv2.imread(os.path.join(all_mias_path, img))
  img_original = cv2.resize(img_original, target_size)
  label = lookUp[img.split('.')[0]]['SEVERITY']
  cv2.imwrite(os.path.join(final_path, label, img), img_original)
```

In [17]:

```
for img in ddsm_benign:
  img_original = cv2.imread(os.path.join(ddsm_path, 'Benign Masses', img))
  img_original = cv2.resize(img_original, target_size)
  cv2.imwrite(os.path.join(final_path, 'B', img), img_original)
```

In [18]:

```
for img in ddsm_malignant:
  img_original = cv2.imread(os.path.join(ddsm_path, 'Malignant Masses', img))
  img_original = cv2.resize(img_original, target_size)
  cv2.imwrite(os.path.join(final_path, 'M', img), img_original)
```

In [19]:

```
for img in inbreast_benign:
  img_original = cv2.imread(os.path.join(inbreast_path, 'Benign Masses', img))
  img_original = cv2.resize(img_original, target_size)
  cv2.imwrite(os.path.join(final_path, 'B', img), img_original)
```

In [20]:

```
for img in inbreast_malignant:
  img_original = cv2.imread(os.path.join(inbreast_path, 'Malignant Masses', img))
```

```
    img_original = cv2.resize(img_original, target_size)
    cv2.imwrite(os.path.join(final_path, 'M', img), img_original)
```

In [21]:

```
print('Total number of normal images: {}'.format(len(os.listdir(os.path.join(final_path, 'N')))))
print('Total number of benign images: {}'.format(len(os.listdir(os.path.join(final_path, 'B')))))
print('Total number of malignant images: {}'.format(len(os.listdir(os.path.join(final_path,
'M')))))
```

```
207
1093
1316
```

In [ ]:

Setup

In [1]:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [2]:

```
%cd /content/gdrive/My Drive/Cornell/2020-2021/Biomedical ML Final Project - Melted
Paper/Datasets/archive/all-mias
```

```
/content/gdrive/.shortcut-targets-by-id/13OShW0589KdYJRWaN9GrhtF2vtwdrUUW/Biomedical ML Final
Project - Melted Paper/Datasets/archive/all-mias
```

Baseline Standard ML Classifiers

k-NN, SVM, Random Forest, Logistic Regression, MIAS

Importing Necessary Libraries

In [4]:

```
import numpy as np
import cv2, os, sys, random, pickle, h5py
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import tensorflow as tf
from tensorflow import keras
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings(action="ignore", message="^internal gelsd")
```

Importing Labels and Images

Info.txt lists the films in the MIAS database and provides specific information about the images.

- 1st col: MIAS database reference number
- 2nd col: Character of background tissue

```
F - Fatty
G - Fatty-glandular
D - Dense-glandular
```

- 3rd col: Class of abnormality present

```
CALC - Calcification
CIRC - Well-defined/circumscribed masses
SPIC - Spiculated masses
MISC - Other, ill-defined masses
ARCH - Architectural distortion
ASYM - Asymmetry
NORM - Normal
```

- 4th col: Severity of abnormality

```
B - Benign
```

```
B - Benign
M - Malignant
N - Normal
```

- 5th, 6th col: (x, y) image coordinates of center of abnormality
- 7th col: Approximate radius (in pixels) of a circle enclosing the abnormality

In [5]:

```
df = pd.read_table('Info.txt', delimiter=' ')
df.SEVERITY = df.SEVERITY.fillna('N')
df = df[df.columns[:-1]]
df.head()
```

Out[5]:

	REFNUM	BG	CLASS	SEVERITY	X	Y	RADIUS
0	mdb001	G	CIRC	B	535.0	425.0	197.0
1	mdb002	G	CIRC	B	522.0	280.0	69.0
2	mdb003	D	NORM	N	NaN	NaN	NaN
3	mdb004	D	NORM	N	NaN	NaN	NaN
4	mdb005	F	CIRC	B	477.0	133.0	30.0

In [6]:

```
# visualizing different classifications
df_grouped = df.groupby(['CLASS','SEVERITY'])[['REFNUM']].count()
df_grouped
```

Out[6]:

		REFNUM
CLASS	SEVERITY	
ARCH	B	9
	M	10
ASYM	B	6
	M	9
CALC	B	15
	M	15
CIRC	B	21
	M	4
MISC	B	7
	M	8
NORM	N	207
SPIC	B	11
	M	8

LabelEncoder can be used to transform non-numerical labels (as long as they are hashable and comparable) to numerical labels.

In [7]:

```
le = LabelEncoder()
for col in ['BG', 'CLASS', 'SEVERITY']:
  df[col] = le.fit_transform(df[col])
df['RADIUS'] = df['RADIUS'].fillna(-0)
df['X'] = df['X'].fillna(-1)
df['Y'] = df['Y'].fillna(-1)
df.head()
```

Out[7]:

	REFNUM	BG	CLASS	SEVERITY	X	Y	RADIUS
0	mdb001	2	3	0	535.0	425.0	197.0
1	mdb002	2	3	0	522.0	280.0	69.0
2	mdb003	0	5	2	-1.0	-1.0	0.0
3	mdb004	0	5	2	-1.0	-1.0	0.0
4	mdb005	1	3	0	477.0	133.0	30.0

In [8]:

```
# Extracting Features
X = df.drop(columns=['REFNUM','SEVERITY'])
X.head()
```

Out[8]:

	BG	CLASS	X	Y	RADIUS
0	2	3	535.0	425.0	197.0
1	2	3	522.0	280.0	69.0
2	0	5	-1.0	-1.0	0.0
3	0	5	-1.0	-1.0	0.0
4	1	3	477.0	133.0	30.0

In [9]:

```
# Creating target values
y = df['SEVERITY'].values
print(y[0:5])
```

```
[0 0 2 2 0]
```

In [10]:

```
le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
print(le_name_mapping)
```

```
{'B': 0, 'M': 1, 'N': 2}
```

In [11]:

```
#split dataset into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1, stratify=y
)
```

K Nearest Neighbors (referenced https://towardsdatascience.com/building-a-k-nearest-neighbors-k-nn-model-with-scikit-learn-51209555453a)

In [12]:

```
from sklearn.neighbors import KNeighborsClassifier
# Create KNN classifier
knn = KNeighborsClassifier(n_neighbors = 3)
# Fit the classifier to the data
knn.fit(X_train,y_train)
```

Out[12]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

In [13]:

```
#show first 5 model predictions on the test data
knn.predict(X_test)[0:5]
```

Out[13]:

```
array([2, 2, 0, 0, 2])
```

In [15]:

```
#check accuracy of our model on the test data
print('k-NN accuracy: {}'.format(knn.score(X_test, y_test)))
```

```
k-NN accuracy: 0.8939393939393939
```

Logistic Regression

In [19]:

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(random_state = 0)
lr.fit(X_train, y_train)
lr.predict(X_test)[0:5]
print('Logistic regression accuracy: {}'.format(lr.score(X_test,y_test)))
```

```
Logistic regression accuracy: 0.8484848484848485
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Support Vector Machine

In [20]:

```
from sklearn.svm import SVC
svc = SVC(kernel='linear', random_state=0)
svc.fit(X_train, y_train)
svc.predict(X_test)[0:5]
print('SVM accuracy: {}'.format(svc.score(X_test,y_test)))
```

```
SVM accuracy: 0.803030303030303
```

Random Forest

In [21]:

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state =0)
rf.fit(X_train, y_train)
rf.predict(X_test)[0:5]
print('Random forest accuracy: {}'.format(rf.score(X_test,y_test)))
```

```
Random forest accuracy: 0.803030303030303
```

In [ ]:

In [1]:

```
from google.colab import drive

drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [9]:

```
%cd /content/gdrive/My Drive/Courses/ECE/ECE 5970/Biomedical ML Final Project - Melted Paper
```

```
/content/gdrive/.shortcut-targets-by-id/13OShW0589KdYJRWaN9GrhtF2vtwdrUUW/Biomedical ML Final
Project - Melted Paper
```

Baseline Transfer Learning

VGG-16, MIAS

In [10]:

```
import os, time, cv2, random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
import skimage as sk
import skimage.transform
from sklearn.metrics import auc, roc_curve
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.utils import class_weight
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.metrics import CategoricalAccuracy
from tensorflow.keras.applications import VGG19, VGG16
from tensorflow.keras.layers import Concatenate, Dense, Dropout, Flatten, Input
from tensorflow.python.keras import Sequential
```

In [11]:

```
df = pd.read_table('Datasets/archive/all-mias/Info.txt', delimiter=' ')
df.SEVERITY = df.SEVERITY.fillna('N')
lookUp = df[['REFNUM', 'SEVERITY']].set_index('REFNUM').T.to_dict()
df.head(5)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: UserWarning: DataFrame columns are
not unique, some columns will be omitted.
  This is separate from the ipykernel package so we can avoid doing imports until
```

Out[11]:

	REFNUM	BG	CLASS	SEVERITY	X	Y	RADIUS	Unnamed: 7
0	mdb001	G	CIRC	B	535.0	425.0	197.0	NaN
1	mdb002	G	CIRC	B	522.0	280.0	69.0	NaN
2	mdb003	D	NORM	N	NaN	NaN	NaN	NaN
3	mdb004	D	NORM	N	NaN	NaN	NaN	NaN
4	mdb005	F	CIRC	B	477.0	133.0	30.0	NaN

In [12]:

```
def get_roi(path, df, le):
  images, labels = [], []
  lookUp = {}
  for row in df.iterrows():
    # Read the image.
    image = cv2.imread(os.path.join(path, str(row[1][0]) + '.jpg'), cv2.IMREAD_GRAYSCALE)
    image = image.reshape((image.shape[0], image.shape[1], 1))
    label = str(row[1][3])

    # If abnormal, crop around the tumor
    x2, y2 = 0, 0
    edge = image.shape[0] # mias is default 1024x1024
    if label != 'N' and str(row[1][4]) != 'nan':
      x, y = int(row[1][4]), int(row[1][5])
      x1 = x - 112
      if x1 < 0:
        x1, x2 = 0, 224
      if x2 != 224:
        x2 = x + 112
        if x2 > edge:
          x1, x2 = edge - 224, edge

      y1 = edge - y - 112
      if y1 < 0:
        y1, y2 = 0, 224
      else:
        y2 = edge - y + 112
        if y2 > edge:
          y1, y2 = edge - 224, edge

    # Normal case: crop around centre of image.
    else:
      x1, x2 = int(edge / 2 - 112), int(edge / 2 + 112)
      y1, y2 = int(edge / 2 - 112), int(edge / 2 + 112)

    images.append(image[y1:y2, x1:x2, :])
    labels.append(label)
    lookUp[str(row[1][0])] = (image[y1:y2, x1:x2, :], label)

  labels = to_categorical(le.fit_transform(labels))
  return images, labels, lookUp
```

In [13]:

```
le = LabelEncoder()
images, labels, img_dict = get_roi('Datasets/archive/all-mias', df, le)
```

In [14]:

```
mdb023 = mpimg.imread('Datasets/archive/all-mias/mdb023.jpg')
img = img_dict['mdb023'][0]
# display images
fig, ax = plt.subplots(1,2)
ax[0].imshow(mdb023, cmap='gray');
ax[0].set_title('Original Image')
ax[0].get_xaxis().set_visible(False)
ax[0].get_yaxis().set_visible(False)
ax[1].imshow(img.reshape((img.shape[0], img.shape[1])), cmap='gray');
ax[1].set_title('Cropped Image')
ax[1].get_xaxis().set_visible(False)
ax[1].get_yaxis().set_visible(False)
plt.show()
```

In [15]:

```
# 60-20-20 train-val-test split
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.20, stratify=labels
, shuffle=True)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, stratify=y_trai
n, shuffle=True)
X_train, X_val, X_test = np.array(X_train), np.array(X_val), np.array(X_test)
y_train, y_val, y_test = np.array(y_train), np.array(y_val), np.array(y_test)
```

In [16]:

```
def calculate_class_weights(y_train, le):
    y_train = le.inverse_transform(np.argmax(y_train, axis=1))
    weights = class_weight.compute_class_weight("balanced", np.unique(y_train), y_train)
    return dict(enumerate(weights))
    # return class_weights
```

In [17]:

```
class_weights = calculate_class_weights(y_train, le)
print(class_weights)
```

```
{0: 1.6097560975609757, 1: 2.0625, 2: 0.528}
```

In [18]:

```
def get_class_count(labels):
  num_classes = len(labels[0])
  counts = np.zeros(num_classes)
  for label in labels:
    for i in range(num_classes):
      counts[i] += label[i]
  return counts.tolist()
```

In [19]:

```
def random_shearing(img):
  tf = sk.transform.AffineTransform(shear=random.uniform(-0.3, 0.3))
  return sk.transform.warp(img, tf, order=1, preserve_range=True, mode='wrap')

def random_noise(img):
  return sk.util.random_noise(img)

def random_rotation(img):
  return sk.transform.rotate(img, random.uniform(-30, 30))

def horizontal_flip(img):
  return img[:, ::-1]
```

In [20]:

```
transformation_functions = {
  'shear': random_shearing,
  'rotate': random_rotation,
  'noise': random_noise,
  'horizontal_flip': horizontal_flip,
}
```

In [21]:

```
def transform_images(img, transforms: dict):
  num_transformations = random.randint(0, len(transforms))
  transformed_image = img
  for i in range(num_transformations):
    key = random.choice(list(transforms))
    transformed_image = transforms[key](img)
```

```
  return transformed_image
```

In [22]:

```
def generate_more_images(images, labels, transformation_functions):
  more_images = images
  more_labels = labels

  class_balance = get_class_count(labels)
  img_to_add = [max(class_balance) - i for i in class_balance]

  for i in range(len(img_to_add)):
    if int(img_to_add[i]) == 0:
      continue
    label = np.zeros(len(img_to_add))
    label[i] = 1
    class_label_indices = [i for i, x in enumerate(labels) if np.array_equal(x, label)]
    class_images = [images[i] for i in class_label_indices]

    for k in range(int(img_to_add[i])):
      transformed_image = transform_images(class_images[k % len(class_images)],
transformation_functions)
      transformed_image = transformed_image.reshape(1, 224, 224, 1)

      more_images = np.append(more_images, transformed_image, axis=0)
      more_labels = np.append(more_labels, label.reshape(1, len(label)), axis=0)
  return more_images, more_labels
```

In [23]:

```
y_train_before = y_train
X_train, y_train = generate_more_images(X_train, y_train, transformation_functions)
```

In [24]:

```
print('Training data size BEFORE augmenting: {}'.format(len(y_train_before)))
print('Training data size AFTER augmenting: {}'.format(len(y_train)))
```

```
Training data size BEFORE augmenting: 198
Training data size AFTER augmenting: 375
```

In [25]:

```
def create_vgg16_model():
  input = Input(shape=(224, 224, 1))
  img_conc = Concatenate()([input, input, input])

  # VGG19 model with pre-trained ImageNet weights.
  model = Sequential()

  # Base convolutional layers
  model.add(VGG16(include_top=False, weights="imagenet", input_tensor=img_conc))

  # FC layers
  model.add(Flatten())
  FC = Sequential(name='FullyConnected')
  FC.add(Dropout(0.2, seed=16, name='Dropout'))
  FC.add(Dense(units=512, activation='relu', name='Dense1'))
  FC.add(Dense(units=64, activation='relu', name='Dense2'))
  FC.add(Dense(3, activation='softmax', kernel_initializer="random_uniform", name='Output'))
  model.add(FC)
  print(model.summary())
  print(FC.summary())
  return model
```

In [26]:

```
model_16 = create_vgg16_model()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
```

```
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 1s 0us/step
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688  
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0         
_________________________________________________________________
FullyConnected (Sequential)  (None, 3)                 12878595  
=================================================================
Total params: 27,593,283
Trainable params: 27,593,283
Non-trainable params: 0
_________________________________________________________________
None
Model: "FullyConnected"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
Dropout (Dropout)            (None, 25088)             0         
_________________________________________________________________
Dense1 (Dense)               (None, 512)               12845568  
_________________________________________________________________
Dense2 (Dense)               (None, 64)                32832     
_________________________________________________________________
Output (Dense)               (None, 3)                 195       
=================================================================
Total params: 12,878,595
Trainable params: 12,878,595
Non-trainable params: 0
_________________________________________________________________
None
```

In [27]:

```
batch_size=20
max_epoch_frozen=20
max_epoch_unfrozen=20

def train_model(model, X_train, X_val, y_train, y_val, class_weights, epoch, batch_size):
  # Only train fully connected layers by freezing CNN layers
  model.layers[0].trainable = False
  compile_model(model, 1e-5)
  history1 = fit_model(model, X_train, X_val, y_train, y_val, class_weights, epoch, batch_size)
  training_results(history1, frozen=True)

  return history1

def compile_model(model, learning_rate):
  model.compile(optimizer=Adam(learning_rate), loss=CategoricalCrossentropy(),
metrics=[CategoricalAccuracy()])

def fit_model(model, X_train, X_val, y_train, y_val, class_weights, epoch, batch_size):
  history = model.fit(
      x=X_train,
      y=y_train,
      class_weight=class_weights,
      batch_size=batch_size,
      steps_per_epoch=len(X_train) // batch_size,
      validation_data=(X_val, y_val),
      validation_steps=len(X_val) // batch_size,
      epochs=epoch,
  )
  return history
```

In [28]:

```
def training_results(history, frozen) -> None:
  fig = plt.figure()
  n = len(history.history["loss"])
  plt.figure()
  plt.plot(np.arange(0, n), history.history["loss"], label="train set")
  plt.plot(np.arange(0, n), history.history["val_loss"], label="validation set")
```

```
plt.title('Training Loss ' + ('(Frozen)' if frozen else '(Unfrozen)'))
plt.legend(loc='upper right')
plt.xlabel('Epochs')
plt.ylabel('Cross Entropy Loss')
plt.savefig('output/training-loss-baseline.png')
plt.show()

fig = plt.figure()
n = len(history.history["loss"])
plt.figure()
plt.plot(np.arange(0, n), history.history["categorical_accuracy"], label="train set")
plt.plot(np.arange(0, n), history.history["val_categorical_accuracy"], label="validation set")
plt.title('Training Accuracy ' + ('(Frozen)' if frozen else '(Unfrozen)'))
plt.legend(loc='upper right')
plt.xlabel('Epochs')
plt.ylabel("Accuracy")
plt.ylim(0, 1)
plt.savefig('output/training-accuracy-baseline.png')
plt.show()
```

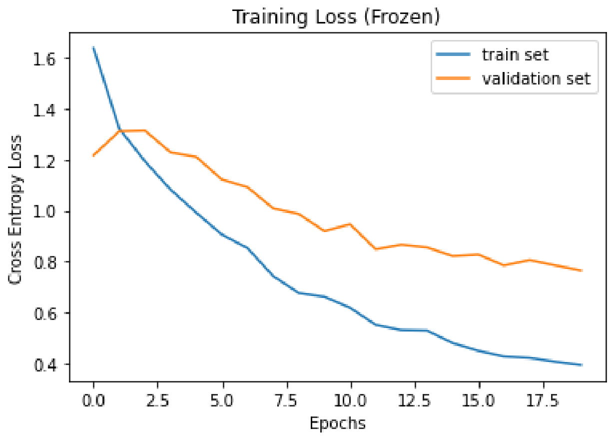
In [29]:

```
batch_size = 20
max_epoch = 20
history_frozen = train_model(model_16, X_train, X_val, y_train, y_val, class_weights, max_epoch, ba
tch_size)
```

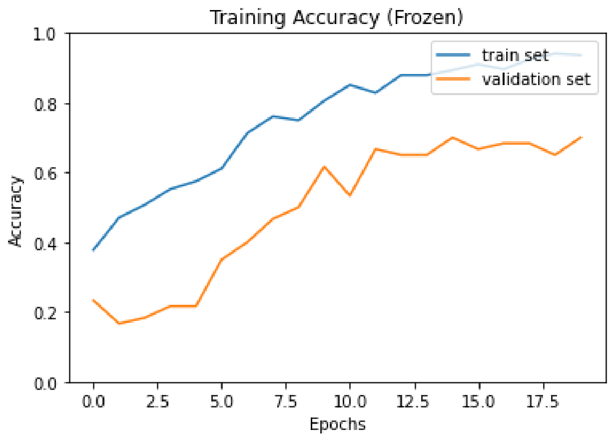
```
Epoch 1/20
18/18 [==============================] - 2s 121ms/step - loss: 1.6364 - categorical_accuracy: 0.37
78 - val_loss: 1.2166 - val_categorical_accuracy: 0.2333
Epoch 2/20
18/18 [==============================] - 2s 102ms/step - loss: 1.3206 - categorical_accuracy: 0.47
04 - val_loss: 1.3106 - val_categorical_accuracy: 0.1667
Epoch 3/20
18/18 [==============================] - 2s 101ms/step - loss: 1.1933 - categorical_accuracy: 0.50
70 - val_loss: 1.3134 - val_categorical_accuracy: 0.1833
Epoch 4/20
18/18 [==============================] - 2s 102ms/step - loss: 1.0819 - categorical_accuracy: 0.55
21 - val_loss: 1.2280 - val_categorical_accuracy: 0.2167
Epoch 5/20
18/18 [==============================] - 2s 101ms/step - loss: 0.9913 - categorical_accuracy: 0.57
46 - val_loss: 1.2103 - val_categorical_accuracy: 0.2167
Epoch 6/20
18/18 [==============================] - 2s 101ms/step - loss: 0.9058 - categorical_accuracy: 0.61
13 - val_loss: 1.1217 - val_categorical_accuracy: 0.3500
Epoch 7/20
18/18 [==============================] - 2s 101ms/step - loss: 0.8524 - categorical_accuracy: 0.71
27 - val_loss: 1.0913 - val_categorical_accuracy: 0.4000
Epoch 8/20
18/18 [==============================] - 2s 102ms/step - loss: 0.7426 - categorical_accuracy: 0.76
06 - val_loss: 1.0083 - val_categorical_accuracy: 0.4667
Epoch 9/20
18/18 [==============================] - 2s 102ms/step - loss: 0.6767 - categorical_accuracy: 0.74
93 - val_loss: 0.9861 - val_categorical_accuracy: 0.5000
Epoch 10/20
18/18 [==============================] - 2s 102ms/step - loss: 0.6618 - categorical_accuracy: 0.80
56 - val_loss: 0.9192 - val_categorical_accuracy: 0.6167
Epoch 11/20
18/18 [==============================] - 2s 102ms/step - loss: 0.6187 - categorical_accuracy: 0.85
07 - val_loss: 0.9462 - val_categorical_accuracy: 0.5333
Epoch 12/20
18/18 [==============================] - 2s 103ms/step - loss: 0.5516 - categorical_accuracy: 0.82
82 - val_loss: 0.8481 - val_categorical_accuracy: 0.6667
Epoch 13/20
18/18 [==============================] - 2s 103ms/step - loss: 0.5309 - categorical_accuracy: 0.87
89 - val_loss: 0.8661 - val_categorical_accuracy: 0.6500
Epoch 14/20
18/18 [==============================] - 2s 103ms/step - loss: 0.5291 - categorical_accuracy: 0.87
89 - val_loss: 0.8553 - val_categorical_accuracy: 0.6500
Epoch 15/20
18/18 [==============================] - 2s 103ms/step - loss: 0.4800 - categorical_accuracy: 0.89
30 - val_loss: 0.8213 - val_categorical_accuracy: 0.7000
Epoch 16/20
18/18 [==============================] - 2s 104ms/step - loss: 0.4500 - categorical_accuracy: 0.90
99 - val_loss: 0.8272 - val_categorical_accuracy: 0.6667
```

```
Epoch 17/20
18/18 [==============================] - 2s 105ms/step - loss: 0.4279 - categorical_accuracy: 0.89
58 - val_loss: 0.7846 - val_categorical_accuracy: 0.6833
Epoch 18/20
18/18 [==============================] - 2s 104ms/step - loss: 0.4222 - categorical_accuracy: 0.92
39 - val_loss: 0.8058 - val_categorical_accuracy: 0.6833
Epoch 19/20
18/18 [==============================] - 2s 104ms/step - loss: 0.4071 - categorical_accuracy: 0.94
08 - val_loss: 0.7848 - val_categorical_accuracy: 0.6500
Epoch 20/20
18/18 [==============================] - 2s 107ms/step - loss: 0.3943 - categorical_accuracy: 0.93
61 - val_loss: 0.7645 - val_categorical_accuracy: 0.7000
```

```
<Figure size 432x288 with 0 Axes>
```




```
<Figure size 432x288 with 0 Axes>
```



In [30]:

```
from sklearn.metrics import accuracy_score, confusion_matrix
def evaluate_model(prediction, y_true, le):
  true_y = le.inverse_transform(np.argmax(y_true, axis=1))
  pred_y = le.inverse_transform(np.argmax(prediction, axis=1))

  # Calculate accuracy
  accuracy = float('{:.4f}'.format(accuracy_score(true_y, pred_y)))
  print("Accuracy = {}\n".format(accuracy))

  cm = confusion_matrix(true_y, pred_y)
  cm_norm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
  cm_norm[np.isnan(cm_norm)] = 0
  plot_confusion_matrix(cm_norm, le)
```

In [31]:

```
def plot_confusion_matrix(cm, le):
  fig, ax = plt.subplots(figsize=(6, 4))
  sns.heatmap(cm, annot=True, ax=ax, fmt='.2f', cmap=plt.cm.Blues, vmin=0, vmax=1)  # annot=True to
```

```
annotate cells

  # Set labels, title, ticks and axis range.
  ax.set_xlabel('Predicted Classes')
  ax.set_ylabel('True Classes')
  ax.set_title('Confusion Matrix')
  ax.xaxis.set_ticklabels(le.classes_)
  ax.yaxis.set_ticklabels(le.classes_)
  plt.tight_layout()
  bottom, top = ax.get_ylim()
  plt.show()
  plt.savefig('output/confusion_mat.png')
```

In [36]:

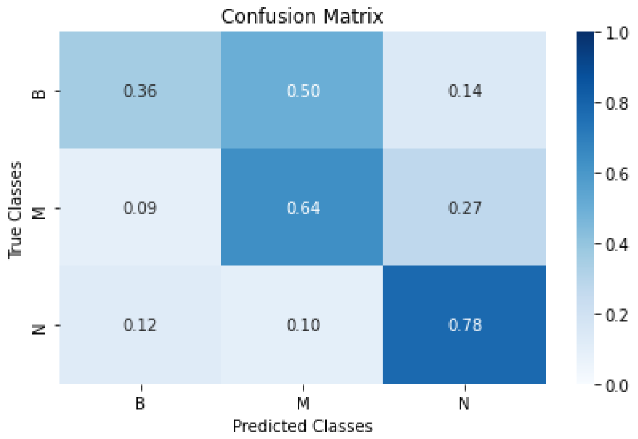
```
def plot_roc(y_true, y_pred, le):
  fpr, tpr, roc_auc = {}, {}, {}

  for i in range(le.classes_.size):
      fpr[i], tpr[i], _ = roc_curve(y_true[:, i], y_pred[:, i])
      roc_auc[i] = auc(fpr[i], tpr[i])

  all_fpr = np.unique(np.concatenate([fpr[i] for i in range(len(le.classes_))]))

  mean_tpr = np.zeros_like(all_fpr)
  for i in range(le.classes_.size):
      mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

  mean_tpr /= le.classes_.size

  fpr['macro'] = all_fpr
  tpr['macro'] = mean_tpr
  roc_auc['macro'] = auc(fpr['macro'], tpr['macro'])

  fpr['micro'], tpr['micro'], _ = roc_curve(y_true.ravel(), y_pred.ravel())
  roc_auc['micro'] = auc(fpr['micro'], tpr['micro'])
  plt.figure(figsize=(8, 5))

  plt.plot([0, 1], [0, 1], 'k--', color='black', lw=2)
  plt.annotate('Random Guess', (.54, .49), color='black')

  plt.plot(fpr['micro'], tpr['micro'],
           label='Micro-Average ROC curve (area = {0:0.2f})'
                 ''.format(roc_auc["micro"]),
           color='red', linestyle=':', lw=3)

  plt.plot(fpr['macro'], tpr['macro'],
           label='Macro-Average ROC curve (area = {0:0.2f})'
                 ''.format(roc_auc['macro']),
           color='black', linestyle=':', lw=3)

  colors = ['#3972ba', '#ab923e', '#3bb300']
  for i, color in zip(range(len(le.classes_)), colors):
      plt.plot(fpr[i], tpr[i], color=color, lw=2,
               label='ROC of Class {0} (area = {1:0.2f})'
                     ''.format(le.classes_[i], roc_auc[i]))

  plt.xlim([0.0, 1.0])
  plt.ylim([0.0, 1.0])
  plt.xlabel('False Positive Rate')
  plt.ylabel('True Positive Rate')
  plt.title('ROC Curve')
  plt.legend(loc='lower right')
  plt.savefig('output/ROC.png')
  plt.show()

  plt.figure(figsize=(8, 5))
  plt.plot(fpr['micro'], tpr['micro'],
           label='Micro-Average ROC curve (area = {0:0.2f})'
                 ''.format(roc_auc["micro"]),
           color='red', linestyle=':', lw=3)

  plt.plot(fpr['macro'], tpr['macro'],
           label='Macro-Average ROC curve (area = {0:0.2f})'
                 ''.format(roc_auc['macro']),
           color='black', linestyle=':', lw=3)
```

```
colors = ['#3972ba', '#ab923e', '#3bb300']
for i, color in zip(range(len(le.classes_)), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label='ROC of Class {0} (area = {1:0.2f})'
                   ''.format(le.classes_[i], roc_auc[i]))

plt.xlim(0, 0.2)
plt.ylim(0.8, 1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (Top Left)')
plt.legend(loc='lower right')
plt.savefig('output/ROC-top-left.png')
plt.show()
```

In [32]:

```
# validation
prediction = model_16.predict(X_val, batch_size=20)
evaluate_model(prediction, y_val, le)
```

Accuracy = 0.6667



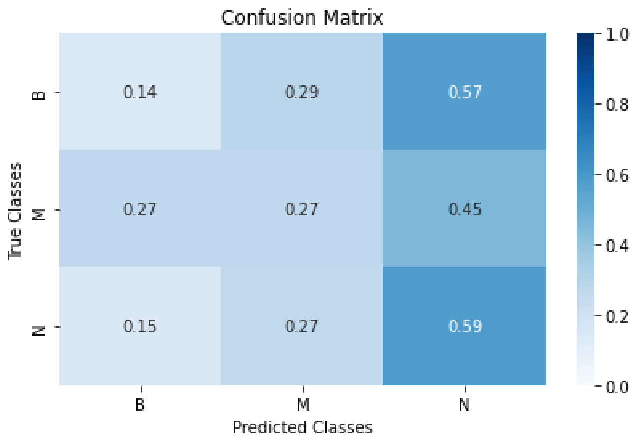
<Figure size 432x288 with 0 Axes>

Testing

In [33]:

```
predictions = model_16.predict(x=X_test)
evaluate_model(prediction, y_test, le)
```

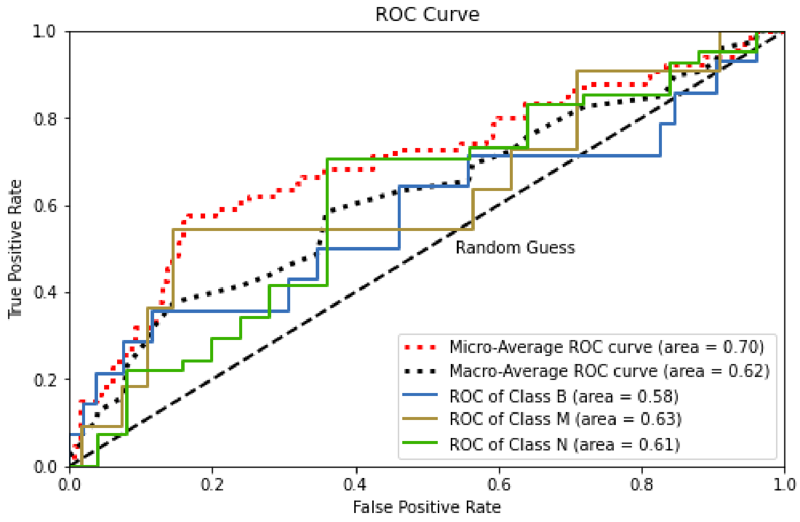
Accuracy = 0.4394

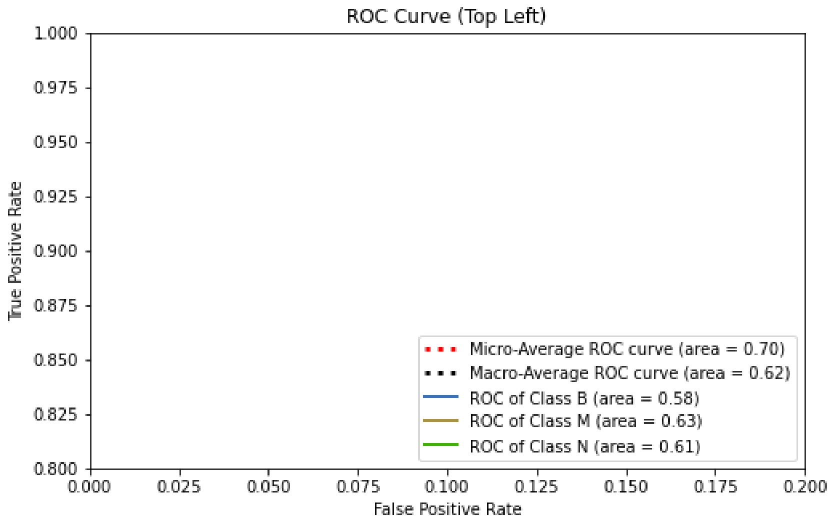


<Figure size 432x288 with 0 Axes>

In [37]:

```
plot_roc(y_test, predictions, le)
```





In [1]:

```
from google.colab import drive

drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [2]:

```
%cd /content/gdrive/My Drive/Courses/ECE/ECE 5970/Biomedical ML Final Project - Melted Paper
```

```
/content/gdrive/My Drive/Courses/ECE/ECE 5970/Biomedical ML Final Project - Melted Paper
```

Final Transfer Learning Model

VGG-16, MIAS, DDSM, INbreast

In [3]:

```
import os, time, cv2, random
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.metrics import auc, roc_curve
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.utils import class_weight
from tensorflow.keras.applications import VGG19, VGG16
from tensorflow.python.keras import Sequential, regularizers
from tensorflow.keras.layers import Concatenate, Dense, Dropout, Flatten, Input
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import CategoricalAccuracy
from tensorflow.keras.utils import to_categorical
```

In [4]:

```
normal_images_path = os.path.join('Datasets', 'final-dataset-augment', 'N')
benign_images_path = os.path.join('Datasets', 'final-dataset-augment', 'B')
malignant_images_path = os.path.join('Datasets', 'final-dataset-augment', 'M')
```

In [5]:

```
normal_images = os.listdir(normal_images_path)
benign_images = os.listdir(benign_images_path)
malignant_images = os.listdir(malignant_images_path)
```

In [6]:

```
images, labels = [], []
```

In [7]:

```
for img in normal_images:
  labels.append('N')
  image = cv2.imread(os.path.join(normal_images_path, img), cv2.IMREAD_GRAYSCALE)
  image = image.reshape((image.shape[0], image.shape[1], 1))
  images.append(image)
```

In [8]:

```
for img in benign_images:
  labels.append('B')
  image = cv2.imread(os.path.join(benign_images_path, img), cv2.IMREAD_GRAYSCALE)
  image = image.reshape((image.shape[0], image.shape[1], 1))
  images.append(image)
```

In [9]:

```
for img in malignant_images:
  labels.append('M')
  image = cv2.imread(os.path.join(malignant_images_path, img), cv2.IMREAD_GRAYSCALE)
  image = image.reshape((image.shape[0], image.shape[1], 1))
  images.append(image)
```

In [10]:

```
le = LabelEncoder()
labels = to_categorical(le.fit_transform(labels))
```

Split Data 60-20-20 for training-validation-testing

In [11]:

```
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.20, stratify=labels
, shuffle=True)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, stratify=y_trai
n, shuffle=True)
X_train, X_val, X_test = np.array(X_train), np.array(X_val), np.array(X_test)
y_train, y_val, y_test = np.array(y_train), np.array(y_val), np.array(y_test)
```

In [12]:

```
print(len(y_train), len(y_test), len(y_val))
```

```
2316 773 773
```

In [13]:

```
def calculate_class_weights(y_train, le):
  y_train = le.inverse_transform(np.argmax(y_train, axis=1))
  weights = class_weight.compute_class_weight("balanced", np.unique(y_train), y_train)
  return dict(enumerate(weights))
  # return class_weights
```

In [14]:

```
class_weights = calculate_class_weights(y_train, le)
print(class_weights)
```

```
{0: 1.0052083333333333, 1: 0.9784537389100126, 2: 1.0171277997364954}
```

VGG16 with L1L2 Regularization

In [15]:

```
def create_vgg16_model():
  input = Input(shape=(224, 224, 1))
  img_conc = Concatenate()([input, input, input])

  # VGG19 model with pre-trained ImageNet weights.
  model = Sequential()

  # Base convolutional layers
  model.add(VGG16(include_top=False, weights="imagenet", input_tensor=img_conc))

  # FC layers
  model.add(Flatten())
```

```
    model.add(Flatten())
    FC = Sequential(name='FullyConnected')
    FC.add(Dense(units=512, kernel_regularizer=regularizers.l1_l2(l1=.001,l2=.001), activation='relu'
, name='Dense1'))
    FC.add(Dense(3, activation='sigmoid', kernel_initializer="random_uniform", name='Output'))
    model.add(FC)
    print(model.summary())
    print(FC.summary())
    return model
```

In [16]:

```
model = create_vgg16_model()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 1s 0us/step
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688  
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0         
_________________________________________________________________
FullyConnected (Sequential)  (None, 3)                 12847107  
=================================================================
Total params: 27,561,795
Trainable params: 27,561,795
Non-trainable params: 0
_________________________________________________________________
None
Model: "FullyConnected"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
Dense1 (Dense)               (None, 512)               12845568  
_________________________________________________________________
Output (Dense)               (None, 3)                 1539      
=================================================================
Total params: 12,847,107
Trainable params: 12,847,107
Non-trainable params: 0
_________________________________________________________________
None
```

In [17]:

```
def train_model(model, X_train, X_val, y_train, y_val, class_weights, epoch, batch_size, title=""):
    # Only train fully connected layers by freezing CNN layers
    model.layers[0].trainable = False
    compile_model(model, 1e-5)
    history1 = fit_model(model, X_train, X_val, y_train, y_val, class_weights, epoch, batch_size)
    training_results(history1, True, title)

    # Unfreeze all layers
    model.layers[0].trainable = True
    compile_model(model, 1e-5)
    history2 = fit_model(model, X_train, X_val, y_train, y_val, class_weights, epoch, batch_size)
    training_results(history2, False, title)

    return history1, history2

def compile_model(model, learning_rate):
    model.compile(optimizer=Adam(learning_rate), loss=CategoricalCrossentropy(),
metrics=[CategoricalAccuracy()])

def fit_model(model, X_train, X_val, y_train, y_val, class_weights, epoch, batch_size):
    history = model.fit(
        x=X_train,
        y=y_train,
        class_weight=class_weights,
        batch_size=batch_size,
        steps_per_epoch=len(X_train) // batch_size,
        validation_data=(X_val, y_val),
```

```
        validation_steps=len(X_val) // batch_size,
        epochs=epoch,
    )
    return history
```

In [18]:

```
def training_results(history, frozen, title):
    fig = plt.figure()
    n = len(history.history["loss"])
    plt.figure()
    plt.plot(np.arange(0, n), history.history["loss"], label="train set")
    plt.plot(np.arange(0, n), history.history["val_loss"], label="validation set")
    plt.title('Training Loss' + (' (Frozen)' if frozen else ' (Unfrozen)'))
    plt.legend(loc='upper right')
    plt.xlabel('Epochs')
    plt.ylabel('Cross Entropy Loss')
    plt.savefig('output/training-loss-' + str(frozen) + '-' + title + '.png')
    plt.show()

    fig = plt.figure()
    n = len(history.history["loss"])
    plt.figure()
    plt.plot(np.arange(0, n), history.history["categorical_accuracy"], label="train set")
    plt.plot(np.arange(0, n), history.history["val_categorical_accuracy"], label="validation set")
    plt.title('Training Accuracy ' + ('(Frozen)' if frozen else '(Unfrozen)'))
    plt.legend(loc='upper right')
    plt.xlabel('Epochs')
    plt.ylabel("Accuracy")
    plt.ylim(0, 1)
    plt.savefig('output/training-accuracy-' + str(frozen) + '-' + title + '.png')
    plt.show()
```

In [19]:

```
from sklearn.metrics import accuracy_score, confusion_matrix
def evaluate_model(prediction, y_true, le, title=""):
    true_y = le.inverse_transform(np.argmax(y_true, axis=1))
    pred_y = le.inverse_transform(np.argmax(prediction, axis=1))
    print(len(pred_y))
    # Calculate accuracy
    accuracy = float('{:.4f}'.format(accuracy_score(true_y, pred_y)))
    print("Accuracy = {}\n".format(accuracy))

    cm = confusion_matrix(true_y, pred_y)
    cm_norm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    cm_norm[np.isnan(cm_norm)] = 0
    plot_confusion_matrix(cm_norm, le, title)
```

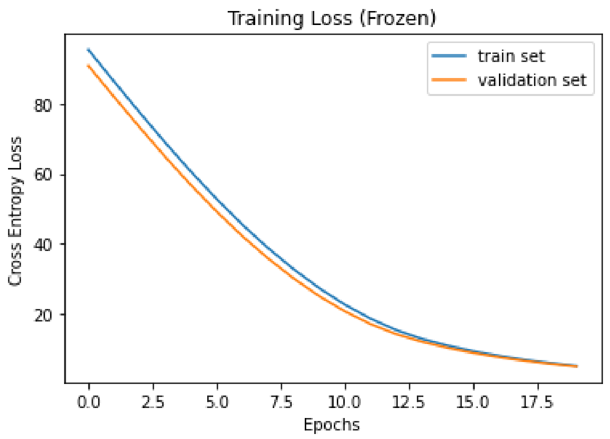
In [20]:

```
batch_size = 20
max_epoch = 20
history_frozen, history_unfrozen = train_model(model, X_train, X_val, y_train, y_val, class_weights
, max_epoch, batch_size, '-l2=0.01')
```

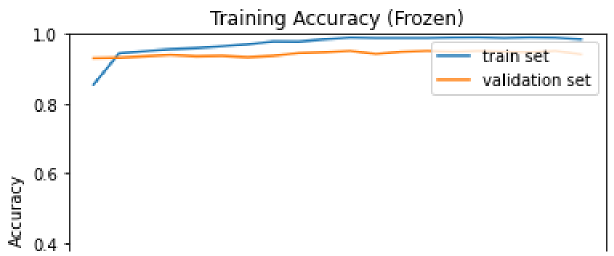
```
Epoch 1/20
115/115 [==============================] - 14s 122ms/step - loss: 95.4280 - categorical_accuracy:
0.8539 - val_loss: 90.8369 - val_categorical_accuracy: 0.9303
Epoch 2/20
115/115 [==============================] - 14s 123ms/step - loss: 86.3343 - categorical_accuracy:
0.9443 - val_loss: 81.9584 - val_categorical_accuracy: 0.9316
Epoch 3/20
115/115 [==============================] - 14s 123ms/step - loss: 77.4560 - categorical_accuracy:
0.9503 - val_loss: 73.1924 - val_categorical_accuracy: 0.9355
Epoch 4/20
115/115 [==============================] - 14s 125ms/step - loss: 68.8088 - categorical_accuracy:
0.9560 - val_loss: 64.7476 - val_categorical_accuracy: 0.9395
Epoch 5/20
115/115 [==============================] - 15s 126ms/step - loss: 60.5226 - categorical_accuracy:
0.9591 - val_loss: 56.6818 - val_categorical_accuracy: 0.9355
Epoch 6/20
115/115 [==============================] - 15s 128ms/step - loss: 52.6966 - categorical_accuracy:
```

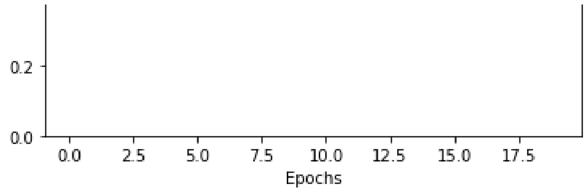
```
0.9643 - val_loss: 49.1168 - val_categorical_accuracy: 0.9368
Epoch 7/20
115/115 [==============================] - 15s 130ms/step - loss: 45.4051 - categorical_accuracy:
0.9699 - val_loss: 42.1260 - val_categorical_accuracy: 0.9329
Epoch 8/20
115/115 [==============================] - 15s 132ms/step - loss: 38.7062 - categorical_accuracy:
0.9782 - val_loss: 35.7482 - val_categorical_accuracy: 0.9368
Epoch 9/20
115/115 [==============================] - 15s 133ms/step - loss: 32.6363 - categorical_accuracy:
0.9778 - val_loss: 29.9990 - val_categorical_accuracy: 0.9447
Epoch 10/20
115/115 [==============================] - 15s 134ms/step - loss: 27.2281 - categorical_accuracy:
0.9843 - val_loss: 24.9445 - val_categorical_accuracy: 0.9474
Epoch 11/20
115/115 [==============================] - 15s 134ms/step - loss: 22.5098 - categorical_accuracy:
0.9891 - val_loss: 20.5992 - val_categorical_accuracy: 0.9513
Epoch 12/20
115/115 [==============================] - 16s 137ms/step - loss: 18.5088 - categorical_accuracy:
0.9878 - val_loss: 16.9545 - val_categorical_accuracy: 0.9421
Epoch 13/20
115/115 [==============================] - 16s 138ms/step - loss: 15.2521 - categorical_accuracy:
0.9878 - val_loss: 14.0905 - val_categorical_accuracy: 0.9487
Epoch 14/20
115/115 [==============================] - 16s 139ms/step - loss: 12.7495 - categorical_accuracy:
0.9878 - val_loss: 11.9355 - val_categorical_accuracy: 0.9513
Epoch 15/20
115/115 [==============================] - 16s 139ms/step - loss: 10.8281 - categorical_accuracy:
0.9891 - val_loss: 10.1889 - val_categorical_accuracy: 0.9487
Epoch 16/20
115/115 [==============================] - 16s 139ms/step - loss: 9.2512 - categorical_accuracy: 0
.9895 - val_loss: 8.7588 - val_categorical_accuracy: 0.9513
Epoch 17/20
115/115 [==============================] - 16s 139ms/step - loss: 7.9262 - categorical_accuracy: 0
.9878 - val_loss: 7.5420 - val_categorical_accuracy: 0.9487
Epoch 18/20
115/115 [==============================] - 16s 139ms/step - loss: 6.8079 - categorical_accuracy: 0
.9895 - val_loss: 6.5113 - val_categorical_accuracy: 0.9474
Epoch 19/20
115/115 [==============================] - 16s 139ms/step - loss: 5.8648 - categorical_accuracy: 0
.9887 - val_loss: 5.6345 - val_categorical_accuracy: 0.9513
Epoch 20/20
115/115 [==============================] - 16s 139ms/step - loss: 5.0551 - categorical_accuracy: 0
.9843 - val_loss: 4.8965 - val_categorical_accuracy: 0.9408
```

```
<Figure size 432x288 with 0 Axes>
```




```
<Figure size 432x288 with 0 Axes>
```

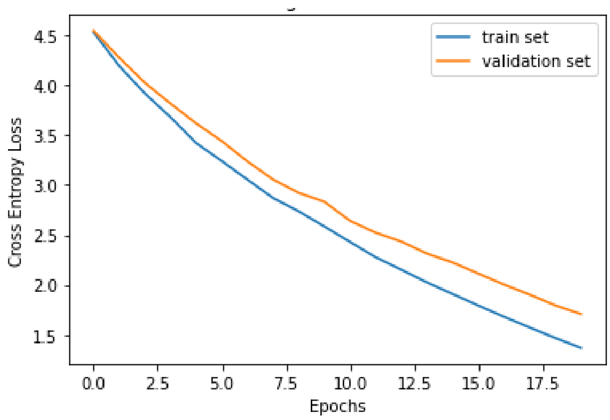




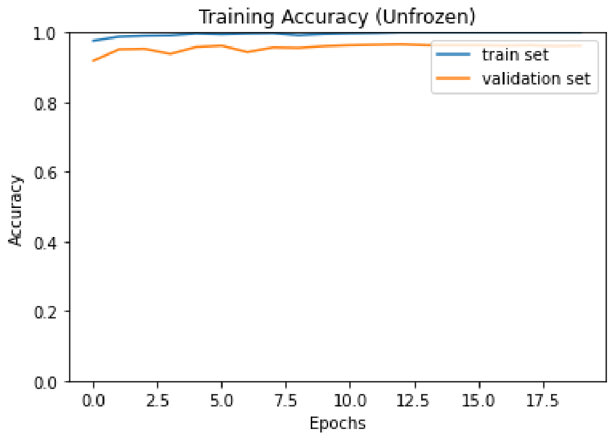

```
Epoch 1/20
  2/115 [..............................] - ETA: 18s - loss: 4.7241 - categorical_accuracy:
1.0000WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the batch time
(batch time: 0.1058s vs `on_train_batch_end` time: 0.2187s). Check your callbacks.
115/115 [==============================] - 40s 347ms/step - loss: 4.5306 - categorical_accuracy: 0
.9761 - val_loss: 4.5458 - val_categorical_accuracy: 0.9184
Epoch 2/20
115/115 [==============================] - 40s 344ms/step - loss: 4.1914 - categorical_accuracy: 0
.9878 - val_loss: 4.2751 - val_categorical_accuracy: 0.9513
Epoch 3/20
115/115 [==============================] - 40s 346ms/step - loss: 3.9180 - categorical_accuracy: 0
.9904 - val_loss: 4.0244 - val_categorical_accuracy: 0.9526
Epoch 4/20
115/115 [==============================] - 40s 345ms/step - loss: 3.6786 - categorical_accuracy: 0
.9913 - val_loss: 3.8157 - val_categorical_accuracy: 0.9382
Epoch 5/20
115/115 [==============================] - 40s 344ms/step - loss: 3.4200 - categorical_accuracy: 0
.9970 - val_loss: 3.6167 - val_categorical_accuracy: 0.9579
Epoch 6/20
115/115 [==============================] - 40s 345ms/step - loss: 3.2401 - categorical_accuracy: 0
.9948 - val_loss: 3.4402 - val_categorical_accuracy: 0.9618
Epoch 7/20
115/115 [==============================] - 40s 345ms/step - loss: 3.0535 - categorical_accuracy: 0
.9970 - val_loss: 3.2359 - val_categorical_accuracy: 0.9434
Epoch 8/20
115/115 [==============================] - 40s 345ms/step - loss: 2.8692 - categorical_accuracy: 0
.9978 - val_loss: 3.0546 - val_categorical_accuracy: 0.9566
Epoch 9/20
115/115 [==============================] - 40s 344ms/step - loss: 2.7367 - categorical_accuracy: 0
.9917 - val_loss: 2.9212 - val_categorical_accuracy: 0.9553
Epoch 10/20
115/115 [==============================] - 40s 346ms/step - loss: 2.5842 - categorical_accuracy: 0
.9956 - val_loss: 2.8321 - val_categorical_accuracy: 0.9605
Epoch 11/20
115/115 [==============================] - 40s 345ms/step - loss: 2.4297 - categorical_accuracy: 0
.9974 - val_loss: 2.6413 - val_categorical_accuracy: 0.9632
Epoch 12/20
115/115 [==============================] - 39s 343ms/step - loss: 2.2757 - categorical_accuracy: 0
.9983 - val_loss: 2.5224 - val_categorical_accuracy: 0.9645
Epoch 13/20
115/115 [==============================] - 39s 342ms/step - loss: 2.1514 - categorical_accuracy: 1
.0000 - val_loss: 2.4331 - val_categorical_accuracy: 0.9658
Epoch 14/20
115/115 [==============================] - 39s 341ms/step - loss: 2.0245 - categorical_accuracy: 1
.0000 - val_loss: 2.3129 - val_categorical_accuracy: 0.9632
Epoch 15/20
115/115 [==============================] - 39s 341ms/step - loss: 1.9081 - categorical_accuracy: 1
.0000 - val_loss: 2.2236 - val_categorical_accuracy: 0.9632
Epoch 16/20
115/115 [==============================] - 39s 340ms/step - loss: 1.7924 - categorical_accuracy: 1
.0000 - val_loss: 2.1131 - val_categorical_accuracy: 0.9632
Epoch 17/20
115/115 [==============================] - 39s 340ms/step - loss: 1.6806 - categorical_accuracy: 1
.0000 - val_loss: 2.0033 - val_categorical_accuracy: 0.9618
Epoch 18/20
115/115 [==============================] - 39s 339ms/step - loss: 1.5729 - categorical_accuracy: 1
.0000 - val_loss: 1.9023 - val_categorical_accuracy: 0.9632
Epoch 19/20
115/115 [==============================] - 39s 339ms/step - loss: 1.4684 - categorical_accuracy: 1
.0000 - val_loss: 1.7943 - val_categorical_accuracy: 0.9605
Epoch 20/20
115/115 [==============================] - 39s 339ms/step - loss: 1.3689 - categorical_accuracy: 1
.0000 - val_loss: 1.7078 - val_categorical_accuracy: 0.9618
```

```
<Figure size 432x288 with 0 Axes>
```

Training Loss (Unfrozen)



<Figure size 432x288 with 0 Axes>



In [21]:

```
def plot_roc(y_true, y_pred, le):
  fpr, tpr, roc_auc = {}, {}, {}

  for i in range(le.classes_.size):
      fpr[i], tpr[i], _ = roc_curve(y_true[:, i], y_pred[:, i])
      roc_auc[i] = auc(fpr[i], tpr[i])

  all_fpr = np.unique(np.concatenate([fpr[i] for i in range(len(le.classes_))]))

  mean_tpr = np.zeros_like(all_fpr)
  for i in range(le.classes_.size):
      mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

  mean_tpr /= le.classes_.size

  fpr['macro'] = all_fpr
  tpr['macro'] = mean_tpr
  roc_auc['macro'] = auc(fpr['macro'], tpr['macro'])

  fpr['micro'], tpr['micro'], _ = roc_curve(y_true.ravel(), y_pred.ravel())
  roc_auc['micro'] = auc(fpr['micro'], tpr['micro'])
  plt.figure(figsize=(8, 5))

  plt.plot([0, 1], [0, 1], 'k--', color='black', lw=2)
  plt.annotate('Random Guess', (.54, .49), color='black')

  plt.plot(fpr['micro'], tpr['micro'],
           label='Micro-Average ROC curve (area = {0:0.2f})'
                 ''.format(roc_auc["micro"]),
           color='red', linestyle=':', lw=3)

  plt.plot(fpr['macro'], tpr['macro'],
           label='Macro-Average ROC curve (area = {0:0.2f})'
                 ''.format(roc_auc['macro']),
           color='black', linestyle=':', lw=3)

  colors = ['#3972ba', '#ab923e', '#3bb300']
```

```
  for i, color in zip(range(len(le.classes_)), colors):
      plt.plot(fpr[i], tpr[i], color=color, lw=2,
                label='ROC of Class {0} (area = {1:0.2f})'
                      ''.format(le.classes_[i], roc_auc[i]))

  plt.xlim([0.0, 1.0])
  plt.ylim([0.0, 1.0])
  plt.xlabel('False Positive Rate')
  plt.ylabel('True Positive Rate')
  plt.title('ROC Curve')
  plt.legend(loc='lower right')
  plt.savefig('output/ROC.png')
  plt.show()

  plt.figure(figsize=(8, 5))
  plt.plot(fpr['micro'], tpr['micro'],
            label='Micro-Average ROC curve (area = {0:0.2f})'
                  ''.format(roc_auc["micro"]),
            color='red', linestyle=':', lw=3)

  plt.plot(fpr['macro'], tpr['macro'],
            label='Macro-Average ROC curve (area = {0:0.2f})'
                  ''.format(roc_auc['macro']),
            color='black', linestyle=':', lw=3)

  colors = ['#3972ba', '#ab923e', '#3bb300']
  for i, color in zip(range(len(le.classes_)), colors):
      plt.plot(fpr[i], tpr[i], color=color, lw=2,
                label='ROC of Class {0} (area = {1:0.2f})'
                      ''.format(le.classes_[i], roc_auc[i]))

  plt.xlim(0, 0.2)
  plt.ylim(0.8, 1)
  plt.xlabel('False Positive Rate')
  plt.ylabel('True Positive Rate')
  plt.title('ROC Curve (Top Left)')
  plt.legend(loc='lower right')
  plt.savefig('output/ROC-top-left.png')
  plt.show()
```

In [22]:

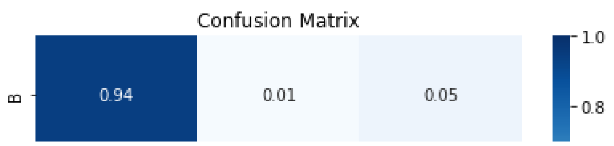
```
def plot_confusion_matrix(cm, le, title=""):
  fig, ax = plt.subplots(figsize=(6, 4))
  sns.heatmap(cm, annot=True, ax=ax, fmt='.2f', cmap=plt.cm.Blues, vmin=0, vmax=1)

  # Set labels, title, ticks and axis range.
  ax.set_xlabel('Predicted Classes')
  ax.set_ylabel('True Classes')
  ax.set_title('Confusion Matrix')
  ax.xaxis.set_ticklabels(le.classes_)
  ax.yaxis.set_ticklabels(le.classes_)
  plt.tight_layout()
  bottom, top = ax.get_ylim()
  plt.show()
  plt.savefig('output/confusion_mat' + title + '.png')
```

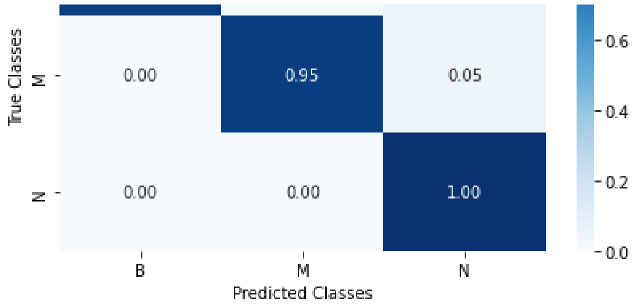
Validation

In [23]:

```
prediction = model.predict(X_val, batch_size=20)
evaluate_model(prediction, y_val, le, '-val-l2=0.01')
```

```
773
Accuracy = 0.9612
```





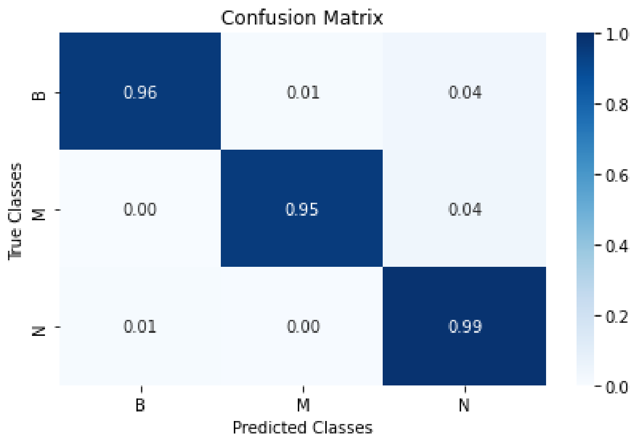

```
<Figure size 432x288 with 0 Axes>
```

Testing

In [24]:

```
predictions = model.predict(x=X_test)
evaluate_model(predictions, y_test, le, '-test-l2=0.01')
```

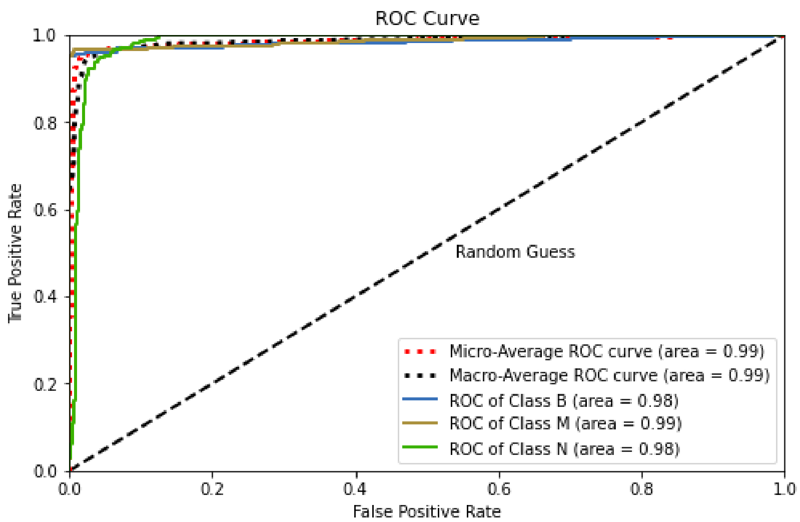
```
773
Accuracy = 0.9664
```

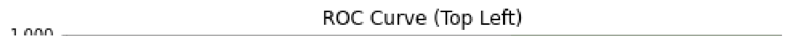


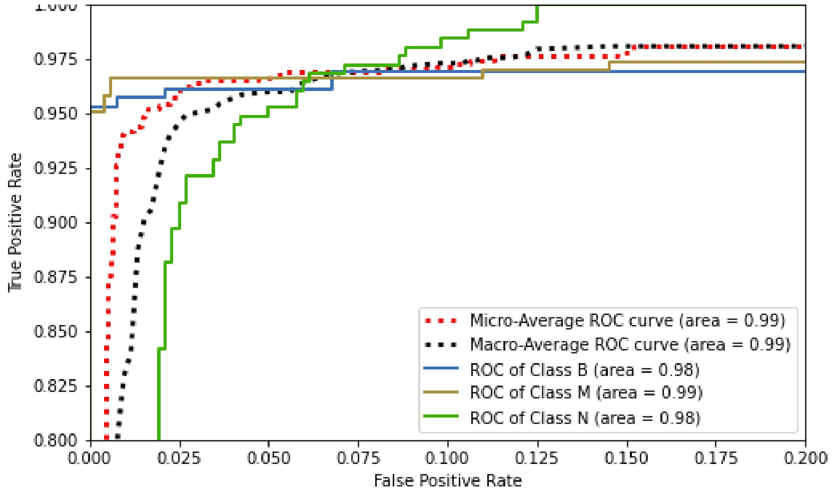

```
<Figure size 432x288 with 0 Axes>
```

In [25]:

```
plot_roc(y_test, predictions, le)
```







1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800
True Positive Rate
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
False Positive Rate
Micro-Average ROC curve (area = 0.99)
Macro-Average ROC curve (area = 0.99)
ROC of Class B (area = 0.98)
ROC of Class M (area = 0.99)
ROC of Class N (area = 0.98)

In [ ]: