Skin Lesion Analysis

Eduardo Silva
up201603135@fe.up.pt
Joana Ramos
up201605017@fe.up.pt
João Agulha
up201607930@fe.up.pt

Faculty of Engineering
University of Porto
Porto, Portugal

Abstract

This document describes a proposed implementation for the second project in the Computer Vision class. Its primary task is to create two distinct architectures for the classification of skin lesions images in two classes (benign and malignant), comparing each other's performance. In this particular case, it was implemented both a Bag of Visual Words based solution and a fine-tuned VGG16 convolutional neural network for this phase. As a secondary task, the previous classification is to be expanded in a wider range of classes: melanoma, melanocytic nevus, basal cell carcinoma, actinic keratosis, benign keratosis, dermatofibroma and vascular lesion. To accomplish this, the previously adapted VGG network was used.

1 Introduction

The broad goal of this project is to develop a system that can perform an automatic classification of skin lesions from dermoscopic images. Dermoscopy is an imaging technique that eliminates the surface reflection of skin. By removing surface reflection, visualization of deeper levels of skin is enhanced.
The project (and data sets used) is based on the ISIC 2016 [1] and 2018 [3] challenges, with part 3 of the 2016 challenge being identical to the first task of this project [2]. As such, there is already a number of solutions and papers published regarding this challenge. In the 2016 challenge, the most successful solution was by Lequan Yu and had an average accuracy of 0.855 and an average precision of 0.637 [2].

2 Proposed Solutions

In this section, both approaches to task 1, bag of visual words and VGG convolutional neural network, are discussed as well as the data sets used for the training and testing of each one, as well as an evaluation of the results produced and some final considerations.

Regarding task 2, the VGG architecture was chosen for its implementation since it produced more favorable results in task 1, therefore all aspects regarding it are also discussed in the respective sub-sections.

2.1 Methodology

Two main architectures were built in accordance to task 1: one following a bag of visual words approach and another based on the well known convolutional neural network for image classification, VGG16. In this sub-section an overview is made regarding each one, describing their implementation.

2.1.1 Bag of Visual Words

The implemented Bag of Visual Words follows the literature provided in the theoretical classes. In more detail, the developed algorithm follows the following steps:

1. Load ground truth of train and test images.
2. Calculate train and test images features with SIFT algorithm.
3. Extract train and test images descriptors from calculated features.
4. Create a bag of words with a dictionary size of 100 and train images descriptors.
5. Generate target variables (bag of words computation between image and provided feature).
6. Train SVM model with generated train variables.
7. Test SVM model with generated test variables.
8. Calculate confusion matrix and statistical estimates.

2.1.2 VGG16

The implemented VGG16 is based on the same network type already provided by TensorFlow / Keras, with a few modifications.

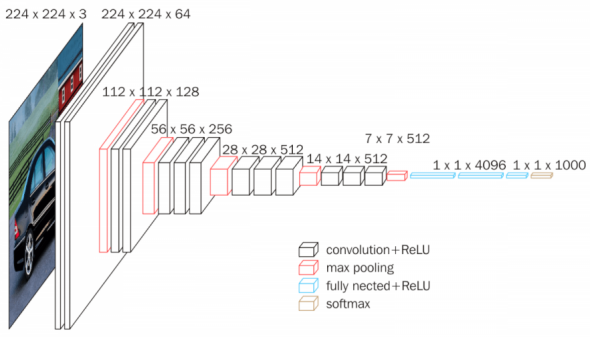

Figure 1: VGG16 Default Architecture

One such customization relates to the input layer, where the input tensor provided to the model is connected to the output of a dedicated VGG16 input pre-processing function.
However, the main difference lies in the last layer, where the number of units depends on the number of classes to classify (2 for task 1 and 7 for task 2), using softmax as an activation function. In addition, since in task 1 the number of training images available wasn't very large, another dropout layer was added before the last one with a dropout rate of 0.3 to avoid overfitting the model.
The number of trainable layers and the pre-existing weights was also changed, with several different models being trained and tested. More specifically, data was gathered regarding the following models:

- M1: All layers trainable and with randomized starting weights.
- M2: All layers trainable and with pre-calculated 'imagenet' weights.
- M3: Only the four last layers (one flattening layer, 2 fully connected dense layers with 4096 units each, the dropout and final dense layer) trainable with pre-calculated 'imagenet' weights.
- M4: Only the last two layers (dropout and the final dense layer) trainable with pre-calculated 'imagenet' weresizeights.

The reasoning behind freezing all the layers except the last 4 or 2 when using pre-calculated weights is due to the nature of convolutional neural networks, where the last layers are more directed towards higher level features, thus leaving the previous, pre-calculated, lower level features layers' weights intact.

The models were then compiled using categorical cross entropy as a loss function (since the input could either be 2 or more classes) with the Adam optimizer. The metrics to display during training are also set here. They will be discussed in the Evaluation sub-section further ahead.

2.2 Training / Testing Data Sets

2.2.1 Bag of Words

The data sets used were the ones provided in the 2016 ISIC challenge [1]. Two different files correspond to the ground truth of the train and test images respectively. It is also possible to specify a lower and upper bound of images so as not to have to train and/or test the model with the whole set of provided images. This filtering is based on the number of each individual image.

2.2.2 VGG16

The data sets used were the ones provided in the 2016 and 2018 ISIC challenges for tasks 1 and 2, respectively. For both of these, the images were split into folders corresponding to their categories so as to ease their manipulation with Tensorflow's / Keras' ImageDataGenerator class. They were also resized to 224 x 224 pixels since that is the original image size the VGG network was trained with. Furthermore, 20% of the images in the training data sets were used for validation during that phase.
This process was applied to task 1's training and testing data sets. However, since task 2's data didn't contain a separate testing set, after splitting the images into the different class folders, 10% of each classes' images were moved to a separate testing set.

2.3 Evaluation

2.3.1 Bag of Words

From the tests carried out with several different architectures, algorithms and parameters it was found that:

1. SIFT performed best for key points calculation.
2. Image resizing and conversion to gray scale helped the model to run faster and perform more accurately and precisely.
3. Bag of words default value of 50 was the acceptable lower end for the dictionary size, but size 100 performed better for the partial and whole data set image classification tasks.
4. SVM worked best with kFold value of 15 for partial and whole data set image classification tasks.

Looking at the results the bag of words model works fairly the same for a training set of images with size 100 or above. Of course, using the whole image data set gives better results overall.

2.3.2 VGG16

From the tests carried out with the several models stated in the Methodology section, it was quickly noted that the most successful ones where those which kept the pre-calculated weights from 'imagenet', training only some of the last layers. These particular models demonstrated a high accuracy in training (> 95%), however, the validation accuracy was lower, around 80% (figures 4 and 7). This might suggest that there was some degree of overfitting, which the test results backed to a certain extent, since there was a high number of malignant cases which were classified as benign (figures 6 and 9). The most likely reason for this is the significantly greater number of benign training images than malignant. Nevertheless, since the number of benign cases in the test set was also substantially higher than malignant ones, it still presented a good accuracy (around 82%) and precision (figures 5 and 8). As expected though, the recall (and consequently F1 score) were lower, around 55% / 60%.
In the cases where all layers were trainable, the training and validation accuracy stayed constant for the most part at around 80%, even over 100 epochs (figures 10 and 13). Test results were far worse however, with all samples being classified as benign, and as such having a low precision (figures 11 and 14). Again, this is most likely due to the imbalance regarding the number of benign and malignant training images.

Regarding Task 2, the results obtained were less satisfactory than the ones regarding Task 1, with training accuracy being similar in the models with only some of the last layers trainable (figures 16 and 19) and slightly lower in the the models with all layers trainable (around 70%) (figures 22 and 25).

The most noticeable difference is in the precision, recall and F1 score values, however. These were slightly lower as well in the models with only the last layers trainable (figures 17 and 20), but in the models with all layers trainable these were significantly worse (figures 23 and 26 26), where once again due to the imbalance in the number of training images, where the number of images belonging to the Melanocytic Nevus class was significantly higher, led these two last models to classify all images as belonging to this class (figures 24 and 27).

3 Conclusion

Overall, the best models for each architecture had a similar performance, with the VGG16 having a slight edge. Although, one may notice that the precision, recall and F1 scores are higher in the Bag of Words architecture (figure 3) than in the VGG (figure 8). This is due to the averaging function used in the VGG (macro averaging), which calculates the metrics for each label, and finds their unweighted mean, not taking label imbalance into account. This means that all classes carry the same weight, even if one of them occurs less frequently (which is the case here). When the results where calculated with micro averaging, the precision, recall and F1 scores had all values closer to 80%.
Besides this, both models also had a tendency to classify (wrongly) some of the images as the most numerous class encountered in training. This could've been improved, however, a major barrier in this type of project lies in the scarcity of time and / or computational resources. Since there are many parameters that can be fine tuned, constructing and testing new combinations of these parameters is a very costly process, both in terms of time and in terms of resources. In addition, the system is always limited to the existing image sets, which immediately creates a bottleneck. As such, future improvements would mostly rely on obtaining a larger data set if possible, otherwise generating "new" images through augmentation of the images in the original sets.

4 Annex

4.1 Task 1

4.2 Bag of Words

```
Confusion Matrix:
[[20  0]
 [ 5  0]]
Accuracy :  80.0 %
Precision:  80.0 %
Recall   :  100.0 %
F1       :  88.89
```

Figure 2: Bag of Words confusion matrix relative to partial data set classification

```
Confusion Matrix:
[[276  25]
 [ 69   5]]
Accuracy :  74.93 %
Precision:  80.0 %
Recall   :  91.69 %
F1       :  85.45
```

Figure 3: Bag of Words confusion matrix relative to full data set classification

4.2.1 VGG16

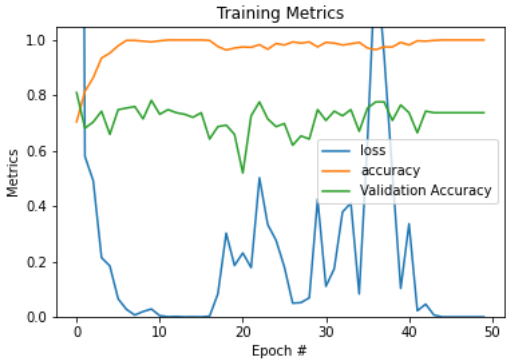

Figure 4: VGG16's model with only the last 2 layers trainable training metrics (imagenet weights)

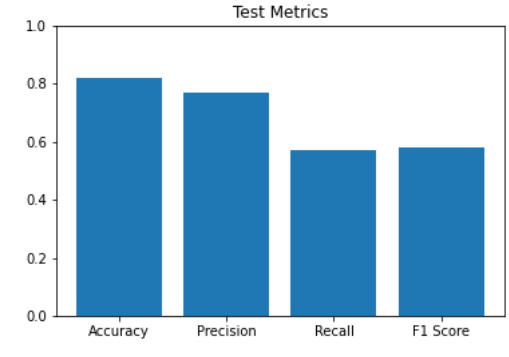

Figure 5: VGG16's model with only the last 2 layers trainable test metrics (imagenet weights)

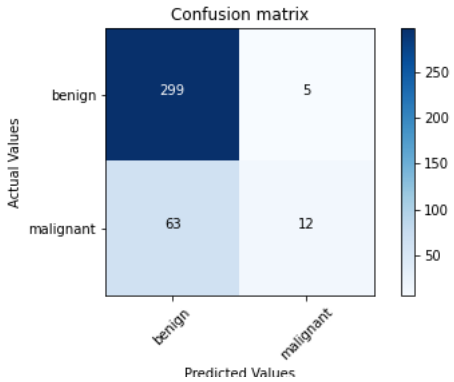

Figure 6: VGG16's model with only the last 2 layers trainable confusion matrix (imagenet weights)

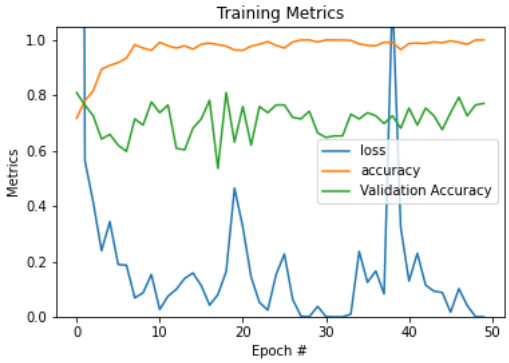

Figure 7: VGG16's model with only the last 5 layers trainable training metrics (imagenet weights)

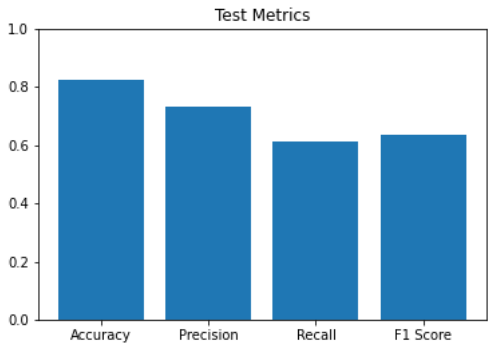

Figure 8: VGG16's model with only the last 5 layers trainable test metrics (imagenet weights)

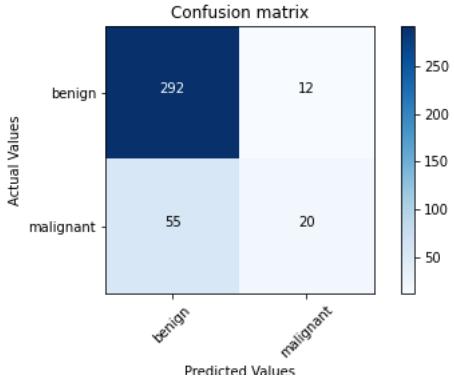

Figure 9: VGG16's model with only the last 5 layers trainable confusion matrix (imagenet weights)

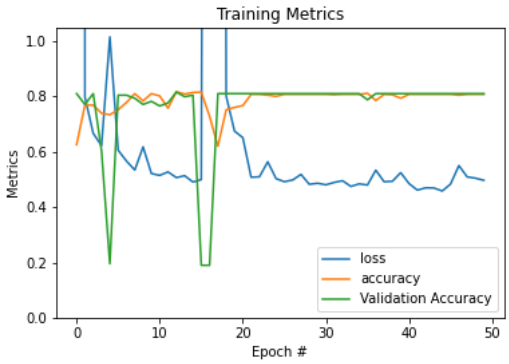

Figure 10: VGG16's model with all layers trainable training metrics (imagenet weights)

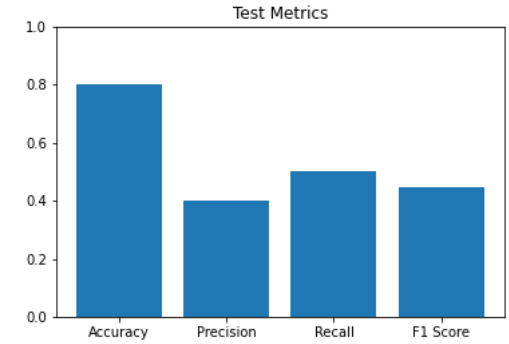

Figure 11: VGG16's model with all layers trainable test metrics (imagenet weights)

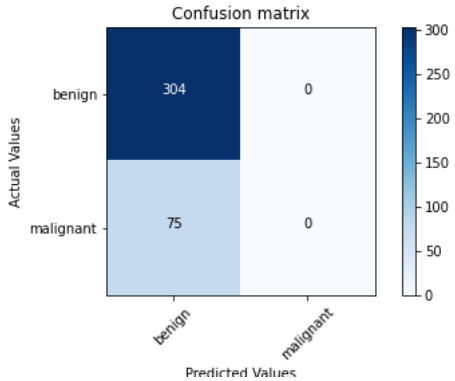

Figure 12: VGG16's model with all layers trainable confusion matrix (imagenet weights)

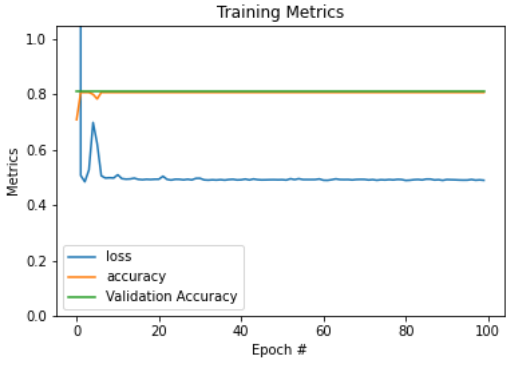

Figure 13: VGG16's model with all layers trainable training metrics (random weights)

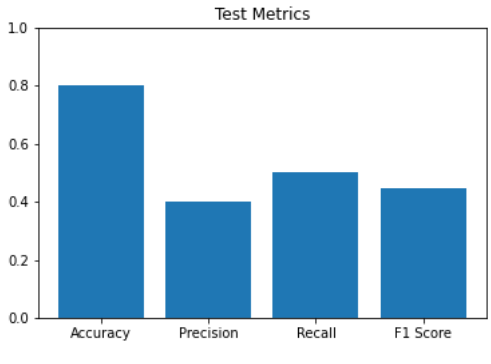

Figure 14: VGG16's model with all layers trainable test metrics (random weights)

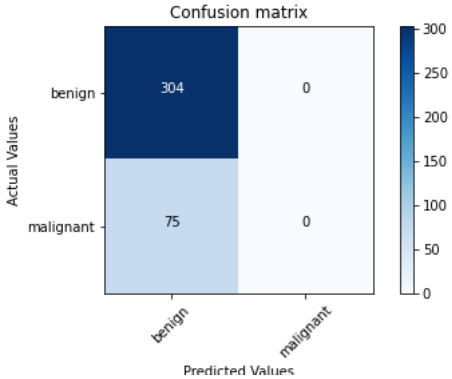

Figure 15: VGG16's model with all layers trainable confusion matrix (random weights)

4.2.2 Task 2

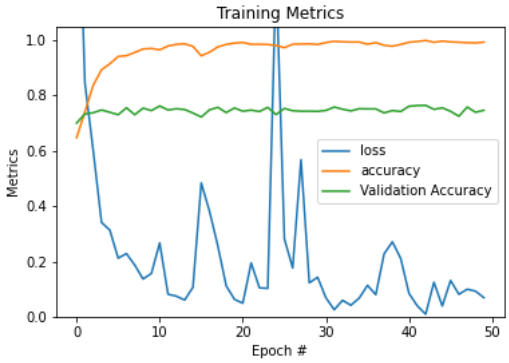

Figure 16: VGG16's model with only the last 2 layers trainable training metrics (imagenet weights), Task 2

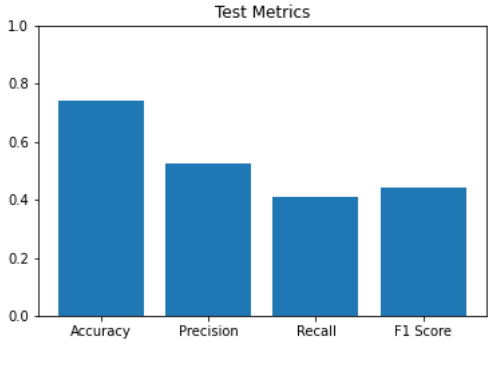

Figure 17: VGG16's model with only the last 2 layers trainable test metrics (imagenet weights), Task 2

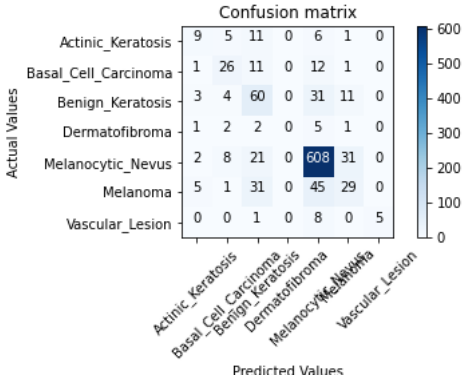

Figure 18: VGG16's model with only the last 2 layers trainable confusion matrix (imagenet weights), Task 2

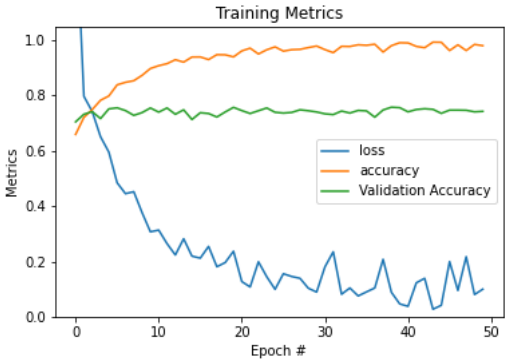

Figure 19: VGG16's model with only the last 5 layers trainable training metrics (imagenet weights), Task 2

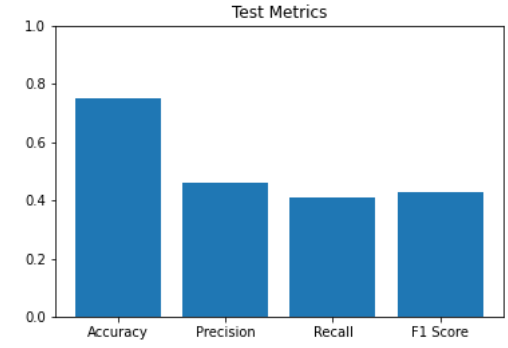

Figure 20: VGG16's model with only the last 5 layers trainable test metrics (imagenet weights), Task 2

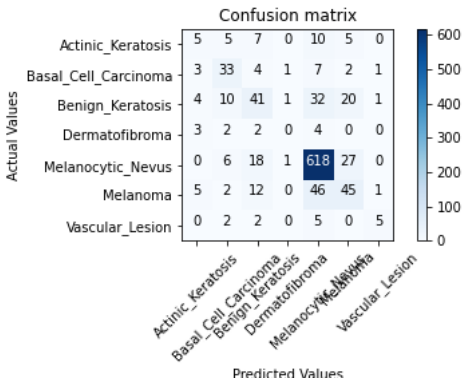

Figure 21: VGG16's model with only the last 5 layers trainable confusion matrix (imagenet weights), Task 2

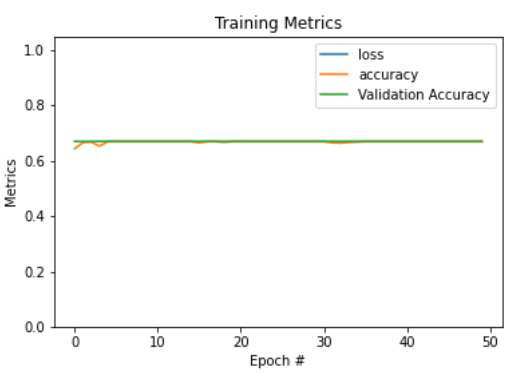

Figure 22: VGG16's model with all layers trainable training metrics (imagenet weights), Task 2

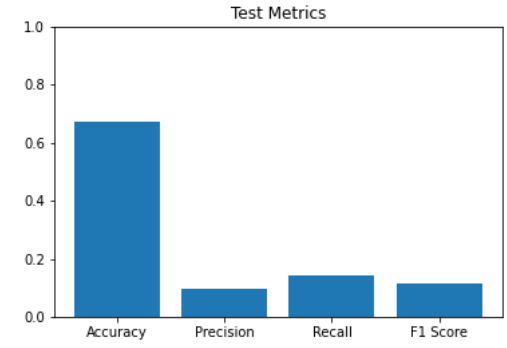

Figure 23: VGG16's model with all layers trainable test metrics (imagenet weights), Task 2

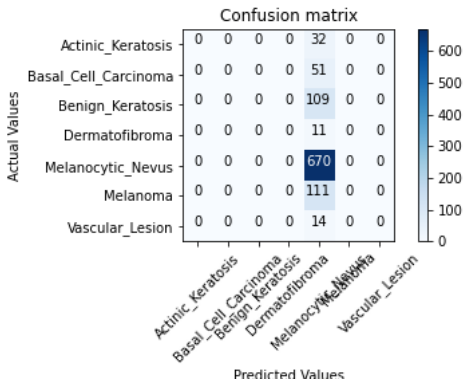

Figure 24: VGG16's model with all layers trainable confusion matrix (imagenet weights), Task 2

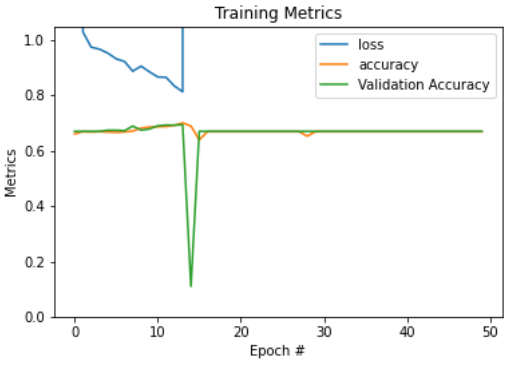

Figure 25: VGG16's model with all layers trainable training metrics (random weights), Task 2

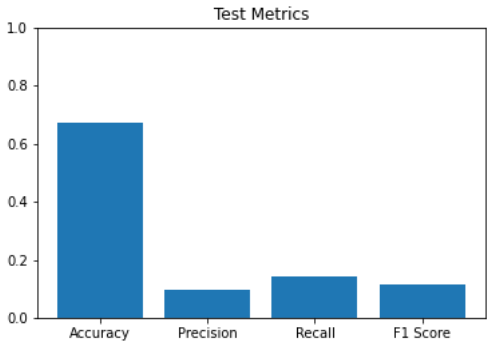

Figure 26: VGG16's model with all layers trainable test metrics (random weights), Task 2

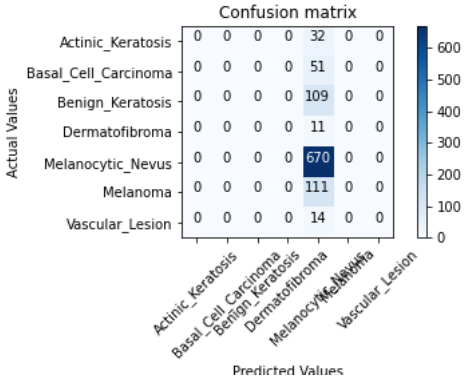

Figure 27: VGG16's model with all layers trainable confusion matrix (random weights), Task 2

4.3 Code

4.3.1 bow.py

```
import sys
import csv
import argparse
from utils import *
from sklearn.metrics import confusion_matrix, f1_score, recall_score, precision_score
sys.path.append('..')


parser = argparse.ArgumentParser(description='VCOM - Bag of Words Tumor classification')
parser.add_argument('-desc', action='store_true', help='Compute image descriptors')
parser.add_argument('-bow', action='store_true', help='Compute Bag of Words')
parser.add_argument('-train', action='store_true', help='Train classifier')
parser.add_argument('-test', action='store_true', help='Test classifier')
args = parser.parse_args()
DESC = args.desc
BOW = args.bow
TRAIN = args.train
TEST = args.test

#-------------------------------------------------------------------------------------------------------------
DESC = 'desc'
BOW = 'bow'
TRAIN = 'train'
TEST = 'test'

#-------------------------------------------------------------------------------------------------------------
IMG_PATH_TRAIN = 'images/train'
IMG_PATH_TEST = 'images/test'


DESC_PATH_TRAIN = 'descriptors_train.pkl'

BOW_PATH = 'bow.pkl'
SVM_TRAIN_PATH = 'model.pkl'
VARS_TRAIN_PATH = 'train.pkl'

db_train = Database(IMG_PATH_TRAIN)
db_test = Database(IMG_PATH_TEST)


labels_train = {
    'lower_bound': 0,
    'upper_bound': 100, #11402
}

labels_test = {
    'lower_bound': 0,
    'upper_bound': 100, #11392
}

keys = {
    'ISIC_0': 0,
    'ISIC_1': 1,
}

def main():
    global db_train
    global db_test

    with open("ISBI2016_ISIC_Part3_Training_GroundTruth.csv") as f:
        keys = {line.split(',')[0]: line.split(',')[1].partition("\n")[0] for line in f}

    with open("TEGT.csv") as f:
        keys.update({line.split(',')[0]: line.split(',')[1].partition("\n")[0] for line in f})

    descriptors_train = []
    features_train = []
    features_test = []
    if not DESC:
        try:
            print('Loading descriptors file from path: ', BOW_PATH)
            descriptors_train = load(DESC_PATH_TRAIN)
            print('Finished loading descriptors')
        except Exception:
            print('Unable to load descriptors file')
            quit()

    print('Starting computing descriptors')
    file_it = labels_train['lower_bound']
    i = 0
    while file_it < labels_train['upper_bound'] :
        try:
            label = 'ISIC_'
            name = label + str(file_it).zfill(7)
            img = db_train.read_img(name)
            img = gray(img)
            img = resize(img)
            feature = calculate_key_points(img)
            features_train.append((name, img, feature))
            if DESC:
                descriptors_train.extend(feature[1])
            print('Computing descriptors for ' + str(label) + str(file_it).zfill(7))
            i = i + 1
```

```
        except:
            continue
        finally:
            file_it = file_it + 1

    file_it = labels_test['lower_bound']
    while file_it < labels_test['upper_bound'] :
        try:
            label = 'ISIC_'
            name = label + str(file_it).zfill(7)
            img = db_test.read_img(name)
            img = gray(img)
            img = resize(img)
            feature = calculate_key_points(img)
            i = i + 1
            features_test.append((name, img, feature))
            print('Computing descriptors for ' + str(label) + str(file_it).zfill(7))
        except:
            continue
        finally:
            file_it = file_it + 1


    print('Finished computing descriptors')
    print('Storing descriptors')
    store(descriptors_train, DESC_PATH_TRAIN)
    print('Finished storing descriptors')

    descriptors_train = to_np(descriptors_train)
    dictionary = None

    if not BOW:
        try:
            print('Loading dictionary file from path: ', BOW_PATH)
            dictionary = load(BOW_PATH)
            print('Finished loading dictionary')
        except:
            print('Unable load dictionary file')
            quit()
    else:
        print('Computing bag of words')
        bow = train_bow(100)
        dictionary = cluster_bow(bow, descriptors_train)
        print('Finished computing bag of words')
        print('Storing bag of words')
        store(dictionary, BOW_PATH)
        print('Finished storing bag of words')

    print('Computing target variables')
    X_train = []
    y_train = []
    X_test = []
    y_test = []
    extractor = bow_retriever(dictionary)
    for name, img, feature in features_train:
        try:
            X_train.append(bow_retrieve(extractor, img, feature[0])[0])
            y_train.append(1 if keys[name] == 'malignant' or keys[name] == '0.0' else 0)
        except:
            continue

    for name, img, feature in features_test:
        try:
            X_test.append(bow_retrieve(extractor, img, feature[0])[0])
            y_test.append(1 if keys[name] == 'malignant' or keys[name] == '0.0' else 0)
        except:
            continue
    print('Finish computing target variables')

    svm = None
    if not TRAIN:
        try:
            print('Loading model file from path: ', SVM_TRAIN_PATH)
            svm = svm_load(SVM_TRAIN_PATH)
            print('Finished loading model')
        except:
            print('Cant load model file')
            quit()
    else:
        print('Training model')
        svm = svm_create()
        svm_train(svm, X_train, y_train)
        print('Finished training model')

        print('Saving model')
        store([X_train,y_train], VARS_TRAIN_PATH)
        svm_store(svm, SVM_TRAIN_PATH)
        print('Finished saving model')


    if TEST:
        print('Testing model')
        pred = np.squeeze(svm_test(svm, X_test)[1].astype(int))
        print('Finished testing model')

        cm = confusion_matrix(y_test,pred)
        print('Confusion Matrix: \n' + str(cm))
        correct = 0
```

```
        for i in range(len(y_test)):
            if y_test[i] == pred[i]:
                correct += 1

        accuracy = correct/len(y_test)*100
        precision = cm[0][0]/(cm[0][0] + cm[1][0])*100
        recall = cm[0][0]/(cm[0][0] + cm[0][1])*100
        f1 = 2 * precision * recall / (precision + recall)

        print('Accuracy : ', round(accuracy, 2), '%')
        print('Precision: ', round(precision, 2), '%')
        print('Recall   : ', round(recall, 2), '%')
        print('F1       : ', round(f1, 2))

if __name__ == "__main__":
    main()
```

4.3.2 utils.py

```
from sklearn import svm
import cv2
import numpy as np
import glob
import pickle
import imutils

WIDTH = 512

def resize(img):
    return imutils.resize(img, width=WIDTH)

def gray(img):
    return cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

def to_np(arr):
    return np.array(arr)

def store(desc_list, path):
    pickle.dump(desc_list, open(path, 'wb'))

def load(path):
    return pickle.load(open(path, 'rb'))

def svm_store(svm, path):
    svm.save(path)

def svm_load(path):
    return cv2.ml.SVM_load(path)

class Database():
    def __init__(self, imgs):
        self.imgs = imgs

    def read_img(self, name):
        return cv2.imread(self._get_img_path(name))

    def _get_img_path(self, name):
        return glob.glob('./'+self.imgs + '/' + name + '.jpg', recursive=True)[0]

def calculate_key_points(img):
    sift = cv2.xfeatures2d.SIFT_create()
    return (sift.detectAndCompute(img, None))

def train_bow(dictionary_size=50):
    return cv2.BOWKMeansTrainer(dictionary_size)

def cluster_bow(trainer, desc):
    return trainer.cluster(desc)

def bow_retrieve(extractor, img, desc):
    return extractor.compute(img, desc)

def bow_retriever(dictionary):
    extractor = cv2.BOWImgDescriptorExtractor(cv2.xfeatures2d.SIFT_create(), cv2.FlannBasedMatcher())
    extractor.setVocabulary(dictionary)
    return extractor

def svm_create():
    svm = cv2.ml.SVM_create()
    svm.setType(cv2.ml.SVM_C_SVC)
    svm.setKernel(cv2.ml.SVM_LINEAR)
    svm.setTermCriteria((cv2.TERM_CRITERIA_MAX_ITER, 100, 1e-6))
    return svm

def svm_train(svm,X,y):
    svm.trainAuto(to_np(X), cv2.ml.ROW_SAMPLE, to_np(y), kFold=15)

def svm_test(svm, pred):
    return svm.predict(np.array(pred))

if __name__ == "__main__":
    pass
```

4.3.3 vgg16.py

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import utils
from imutils import paths
from sklearn.metrics import f1_score, precision_score, recall_score, confusion_matrix, roc_curve, auc, accuracy_score,
                                                                precision_recall_fscore_support
import itertools
import matplotlib.pyplot as plt
import pathlib
import math
import numpy as np
import tensorflow as tf
import cv2
import os

AUTOTUNE = tf.data.experimental.AUTOTUNE
TRAIN_DATA_DIR = '/content/drive/My Drive/Colab Notebooks/data/task2/training_sorted_resized'
TEST_DATA_DIR = '/content/drive/My Drive/Colab Notebooks/data/task2/test_sorted_resized'
IMG_WIDTH = 224
IMG_HEIGHT = 224
BATCH_SIZE = 30
EPOCHS = 50
NAME = 'T2_NONE_ALL'

def load_dataset(train=True):
    print("[INFO] Loading images")

    if train:
      image_generator = tf.keras.preprocessing.image.ImageDataGenerator(validation_split=0.2)
      train_gen = image_generator.flow_from_directory(directory=TRAIN_DATA_DIR, color_mode='rgb', batch_size=BATCH_SIZE,
                                                      shuffle=True, target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                      class_mode='categorical', subset='training')
      valid_gen = image_generator.flow_from_directory(directory=TRAIN_DATA_DIR, color_mode='rgb', batch_size=BATCH_SIZE,
                                                      shuffle=True, target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                      class_mode='categorical', subset='validation')
      return train_gen, valid_gen
    else:
      image_generator = tf.keras.preprocessing.image.ImageDataGenerator()
      test_gen = image_generator.flow_from_directory(directory=TEST_DATA_DIR, color_mode='rgb', batch_size=BATCH_SIZE, shuffle=
                                                      True, target_size=(IMG_HEIGHT, IMG_WIDTH), class_mode='
                                                      categorical')
      return test_gen


# plots images with labels
def plots(ims, figsize=(24,12), rows=4, interp=False, titles=None, predictions=None, keys=None):
    if type(ims[0]) is np.ndarray:
        ims = np.array(ims).astype(np.uint8)
        if (ims.shape[-1] != 3):
            ims = ims.transpose((0,2,3,1))
    f = plt.figure(figsize=figsize)
    cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
    for i in range(len(ims)):
        sp = f.add_subplot(rows, cols, i+1)
        sp.axis('Off')
        if titles is not None:
            title = ''
            prediction = ''

            if(titles is not None):
              if(keys is not None):
                title = keys[titles[i]]
              else:
                title = titles[i]

            title = 'Value: ' + str(title)

            if predictions is not None:
              if keys is not None:
                prediction = keys[predictions[i]]
              else:
                prediction = predictions[i]

              title = title + ' | Prediction: ' + str(prediction)

            sp.set_title(title, fontsize=18)
            sp.set
        plt.imshow(ims[i], interpolation=None if interp else 'none')

def build_model(class_no):
    print("[INFO] Building model")
    input_layer = tf.keras.Input(shape=(IMG_HEIGHT, IMG_WIDTH, 3))
    input_tensor = tf.keras.applications.vgg16.preprocess_input(input_layer)
    baseModel = VGG16(weights=None, include_top=True, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3), input_tensor=input_tensor)

    model = tf.keras.Sequential()

    for layer in baseModel.layers[:-1]:
```

```
      layer.trainable = False
      model.add(layer)

    layer_no = len(model.layers) - 1

    for i in range(layer_no, layer_no - 5, -1):
      model.layers[i].trainable = True

    model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.Dense(class_no, activation='softmax'))
    model.summary()

    return model

def compile_model(model):
  model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=['accuracy', tf.keras.metrics.Precision(), tf.keras.
                                                                    metrics.Recall()])

def train_model(model, train_gen, val_gen):
    print("[INFO] Training model")
    compile_model(model)

    H = model.fit(x=train_gen, validation_data=val_gen, batch_size=BATCH_SIZE, validation_batch_size=BATCH_SIZE, epochs=EPOCHS,
                                                                        verbose=1)
    return model, H

def test_model(model, test_gen, compile=False):
  print("[INFO] Testing model")

  if compile:
    compile_model(model)

  predictions = model.predict(x=test_gen, batch_size=BATCH_SIZE, verbose=1)

  return predictions

def get_generator_data(gen, batch_size):
  examples = len(gen.filenames)
  gen_calls = int(math.ceil(examples / (1.0 * batch_size)))

  images = []
  labels = []

  for i in range(0, gen_calls):
    images.extend(np.array(gen[i][0]))
    labels.extend(np.array(gen[i][1]))

  return images, labels


def plot_confusion_matrix(cm, classes):
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title('Confusion matrix')
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    thresh = cm.max() / 2.

    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Actual Values')
    plt.xlabel('Predicted Values')
    plt.savefig('/content/drive/My Drive/Colab Notebooks/' + NAME + '_cm.png')

train_gen, valid_gen = load_dataset(True)
classes = list(train_gen.class_indices.keys())

imgs, labels = next(train_gen)

if BATCH_SIZE <= 28:
  plots(imgs, titles=np.argmax(labels, axis=1), keys=classes)

model = build_model(len(classes))
model, history = train_model(model, train_gen, valid_gen)
model.save('/content/drive/My Drive/Colab Notebooks/' + NAME)

plt.plot(np.arange(0, EPOCHS), history.history["loss"], label="loss")
plt.plot(np.arange(0, EPOCHS), history.history["accuracy"], label="accuracy")
plt.plot(np.arange(0, EPOCHS), history.history["val_accuracy"], label="Validation Accuracy")
plt.ylim(0, 1.05)
plt.title("Training Metrics")
plt.xlabel("Epoch #")
plt.ylabel("Metrics")
plt.legend(loc="best")
plt.savefig('/content/drive/My Drive/Colab Notebooks/' + NAME + '_plot.png')

test_gen = load_dataset(False)

test_images, test_labels = get_generator_data(test_gen, BATCH_SIZE)
print(test_labels)

predictions = test_model(model, test_gen)
```

```
round_predictions = np.argmax(predictions, axis=1)
round_labels = np.argmax(test_labels, axis=1)
print(round_predictions)
print(round_labels)

confusion_mtx = confusion_matrix(round_labels, round_predictions)
print(str(confusion_mtx))
plot_confusion_matrix(confusion_mtx, classes)

if BATCH_SIZE <= 28:
  plots(test_images[0:9], titles=round_labels, predictions=round_predictions, keys=classes)

test_acc = accuracy_score(round_labels, round_predictions)
test_prec = precision_score(round_labels, round_predictions, average="macro")
test_rec = recall_score(round_labels, round_predictions, average="macro")
test_f1 = f1_score(round_labels, round_predictions, average="macro")

print("Accuracy: ", test_acc)
print("Precision: ", test_prec)
print('Recall: ', test_rec)
print('F1 Score: ', test_f1)

pos = np.arange(4)
plt.bar(pos, [test_acc, test_prec, test_rec, test_f1], align='center')
plt.xticks(pos, ['Accuracy', 'Precision', 'Recall', 'F1 Score'])
plt.title('Test Metrics')
plt.ylim(0, 1)
plt.savefig('/content/drive/My Drive/Colab Notebooks/' + NAME + '_test_metrics.png')
```

4.3.4 group-images.py

```
import os
import csv
import cv2 as cv

print('Run with Python3')

training_input_path = input('Training images folder path (must end with /): ')
csv_train_path = input('Training ground truth table .csv file path: ')
test_input_path = input('Test images folder path (must end with /): ')
csv_test_path = input('Test ground truth table .csv file path: ')

output_path = '../data/task1/training_sorted_resized'
output_path_benign = output_path + '/benign'
output_path_malignant = output_path + '/malignant'

os.makedirs(output_path)
os.makedirs(output_path_benign)
os.makedirs(output_path_malignant)

with open(csv_train_path) as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')

    for row in csv_reader:
        img_name = '/' + row[0] + '.jpg'
        img_input_path = training_input_path + img_name
        img = cv.imread(img_input_path)
        resized = cv.resize(img, (224, 224), interpolation = cv.INTER_AREA)

        classification = row[1]

        if(classification == 'benign'):
            cv.imwrite(output_path_benign + img_name, resized)
        else:
            cv.imwrite(output_path_malignant + img_name, resized)

output_path = '../data/task1/test_sorted_resized'
output_path_benign = output_path + '/benign'
output_path_malignant = output_path + '/malignant'

os.makedirs(output_path)
os.makedirs(output_path_benign)
os.makedirs(output_path_malignant)

with open(csv_test_path) as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')

    for row in csv_reader:
        img_name = '/' + row[0] + '.jpg'
        img_input_path = test_input_path + img_name
        img = cv.imread(img_input_path)

        resized = cv.resize(img, (224, 224), interpolation = cv.INTER_AREA)

        classification = row[1]

        if(classification == '0.0'):
            cv.imwrite(output_path_benign + img_name, resized)
        else:
            cv.imwrite(output_path_malignant + img_name, resized)
```

4.3.5 group-images-task-2.py

```
import os
import csv
import shutil
import cv2 as cv
import math

TEST_FRACTION = 0.1

images_path = input('Training images folder path (must end with /): ')
csv_path = input('Training ground truth table .csv file path: ')

train_output_path = '../data/task2/training_sorted_resized'
test_outputh_path = '../data/task2/testing_sorted_resized'

os.makedirs(train_output_path)
os.makedirs(test_outputh_path)

classes = ['Melanoma', 'Melanocytic_Nevus', 'Basal_Cell_Carcinoma', 'Actinic_Keratosis', 'Benign_Keratosis', 'Dermatofibroma',
                                                                'Vascular_Lesion']

for class_name in classes:
    os.makedirs(train_output_path + '/' + class_name)
    os.makedirs(test_outputh_path + '/' + class_name)

with open(csv_path) as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')

    csv_reader.__next__()

    for row in csv_reader:
        img_name = row[0] + '.jpg'

        img_path = images_path + img_name
        img = cv.imread(img_path)
        print(img_path)
        resized_img = cv.resize(img, (224, 224), interpolation = cv.INTER_AREA)

        j = 1

        for j in range(1, len(row)):
            if row[j] == '1.0':
                classification = row[1]
                break

        print(train_output_path + '/' + classes[j - 1])
        cv.imwrite(train_output_path + '/' + classes[j - 1] + '/' + img_name, resized_img)

for directory in os.listdir(train_output_path):
    train_sub_directory = os.path.join(train_output_path, directory)
    test_sub_directory = os.path.join(test_outputh_path, directory)

    for _, _, images in os.walk(train_sub_directory):
        total_images = len(images)
        test_images_no = math.floor(total_images * TEST_FRACTION)

        for i in range(test_images_no):
            shutil.move(os.path.join(train_sub_directory, images[i]), os.path.join(test_sub_directory, images[i]))
```

References

[1] International Skin Imaging Collaboration
ISIC 2016: Skin Lesion Analysis Towards Melanoma Detection
https://challenge.kitware.com/#challenge/560d7856cad3a57cfde481ba
Accessed in June, 2020.

[2] International Skin Imaging Collaboration
ISIC 2016: Skin Lesion Analysis Towards Melanoma Detection, Part 3: Lesion Classification
https://challenge.kitware.com/#phase/5667455bcad3a56fac786791
Accessed in June, 2020.

[3] International Skin Imaging Collaboration
ISIC 2018: Skin Lesion Analysis Towards Melanoma Detection
https://challenge2018.isic-archive.com/
Accessed in June, 2020.