Machine Learning Project Report

CS545 Machine Learning

"Skin cancer detection and classification using Machine Learning"

Project Members:

Achyutha Harish

PSU ID: 945844917

Deepa Subray Hegde

PSU ID: 954680127

Varsha Karinje

PSU ID: 925923534

Varun Jaisundar Raju

PSU ID: 957710312

GITHUB Link: https://github.com/varunvjr/CS-545-Machine-Learning-Project

Problem Statement and Motivation:

The skin cancer is found in people of age 40-60 years and in males rather than females. To cure the skin cancer, surgery, radiation and photodynamic therapy are alternative ways, while surgery including Mohs microsurgery, laser surgery, and electro desiccation and curettage currently becomes acceptable as effective way with less pain. However, surgery basically depends on skillful medical doctors whose number is quite limited, and it is normally costly so that an automatic system of skin-cancer surgery is really required in order to reliably cure the patients as assistant of medical doctor. Detection and diagnosis of skin cancer lies in the most crucial and tactically the hardest part of the problem. In the entire system we discussed above, we are concentrating on the detection part in this algorithm.

Melanoma Skin cancer is one of the deadly diseases with mortality rate up to 80%. The increase in pollution levels and climatic effect on human skin is increasing the cases of skin cancer and related problems. However, no cure for melanoma has been yet discovered other than cutting out the part of the dermis affected. This method of treatment sometimes causes excess or unaffected region to be damaged. Automatic computer system for overall treatment of skin cancer is still not developed and our project would be effectively detecting the cancerous region and save the unaffected part.

Considering and analyzing features of cancer image, which includes Asymmetry, Border Irregularity, Compact Index, Edge abruptness, Color variation, Diameter and Fractal Dimension. Image segmentation and machine learning concepts are used to detect cancerous cells and distinguish between Benign and Melanoma.

Even though many algorithms have been developed previously, they have failed to provide maximum possible positive results through the database available online. In our opinion, image quality and training dataset space plays a vital role as the limitation has been evident as well. Recent research also shows that these models have outperformed the previous models and one can't simply ignore them due to the results produced are mesmerizing but we are going to prove the predictions of 82-90% in our revised algorithm.

Paper: 1. **Skin cancer** classification **using convolutional neural networks**: systematic review
2. Early **detection** of **skin cancer using** deep learning architectures: **resnet**-101 and inception-v3

Algorithms:

We have used the CNN model to train and test our data. Wide range of algorithms have been tested to provide a reasonable output for this. Algorithms have ranged between 65% to 80%. To get into a bit of introduction to CNN and where it is relevant to our project, CNNs are supervised learning methods, therefore they are trained with data labeled with the appropriate classifications. CNNs, in essence, learn the link between the input objects and the class labels and are composed of two components: the hidden layers that extract the features and, at the conclusion of the processing, the fully connected layers that do the actual classification operation. The hidden layers of a CNN, unlike normal neural networks, have a distinct architecture. In conventional neural networks, each layer is made up of a group of neurons, and each neuron in a layer is linked to every neuron in the layer before it. The architecture of hidden layers in a CNN differs slightly. The neurons in a layer are not linked to all of the neurons in the preceding layer, but rather to a subset of them. This limitation to local connections, along with additional pooling layers that aggregate local neuron outputs into a single value, resulting in translation-invariant features. As a result, the training approach is simplified and the model complexity is reduced. We are making use of ResNet50, a 50 layer deep CNN in form of Keras to increase our model accuracy and implement it as well.

<u>Implementation:</u>

The entire project was built in Anaconda Jupyter Notebook, to break down the project cell by cell:

The first step was to find the right packages, here we have the CUDA toolkit by NVIDIA which is very well renowned in the Image processing and image feature extraction. Other packages were to take TensorFlow for GPU utilization and Keras built on TensorFlow for deep learning algorithms like this one.

We first had to install the right set of packages before importing them and here are the right commands to install them on Anaconda.

```
conda install -c conda-forge keras
```

This command gets us the keras version 2.9.0 installed on our environment. Next let us get and/or update the TensorFlow.

```
conda install -c conda-forge tensorflow
```

We can install the latest TensorFlow 1.14.0 with this command and you will be installing a bunch of different packages along with it. Those packages might not be of use to us in this experiment so let us intentionally skip that one.

The most important part of any Machine Learning algorithm is having clean dataset to train and relevant data to test the model. We are going to use the processed data images from the www.isic-archives.com version 5. We have 1800 Melanoma and 1800 Benign skin lesion and all the images are resized and processed to 244x244.

Now, let us import the packages, first the most generic and common packages that we use for any Machine learning project.

```
import os

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
from glob import glob
import seaborn as sns
from PIL import Image
np.random.seed(29)
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, KFold, cross_val_score, Gri
dSearchCV
from sklearn.metrics import accuracy_score
import itertools
```

```
import keras
from keras.utils.np_utils import to_categorical
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras import backend as K
from keras.layers.normalization import BatchNormalization
from keras.optimizers import Adam, RMSprop
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
from keras.wrappers.scikit_learn import KerasClassifier
from keras.applications.resnet50 import ResNet50
from keras import backend as K
```

Above, we can see that we are using wide range of keras classes to support our project and back our findings. Most of the keras classes are self-explanatory, all of our findings in this import section comes from https://keras.io/ .

Upcoming step involves loading the data and labelling them. In this step, load the images and convert them to numpy arrays using their RGB values. There's no need to resize the images because they've already been scaled to 224x224, as discussed above. Because the images do not have labels, they must be produced. Finally, the images are shuffled together into a large training set.

```
folder_benign_train = "C:\\Users\\shrey\\Downloads\\archive(1)\\data\\train\\benign"
folder_malignant_train = "C:\\Users\\shrey\\Downloads\\archive(1)\\data\\train\\mal
ignant"

folder_benign_test = "C:\\Users\\shrey\\Downloads\\archive(1)\\data\\test\\benign"
folder_malignant_test = "C:\\Users\\shrey\\Downloads\\archive(1)\\data\\test\\malig
nant"

read = lambda imname: np.asarray(Image.open(imname).convert("RGB"))
```

Loading the training images first, followed by testing images,

```
ims_benign = [read(os.path.join(folder_benign_train, filename)) for filename in o
s.listdir(folder_benign_train)]
X_benign = np.array(ims_benign, dtype='uint8')
ims_malignant = [read(os.path.join(folder_malignant_train, filename)) for filenam
e in os.listdir(folder_malignant_train)]
X_malignant = np.array(ims_malignant, dtype='uint8')
ims_benign = [read(os.path.join(folder_benign_test, filename)) for filename in os
.listdir(folder_benign_test)]
X_benign_test = np.array(ims_benign, dtype='uint8')
ims_malignant = [read(os.path.join(folder_malignant_test, filename)) for filename
in os.listdir(folder_malignant_test)]
X_malignant_test = np.array(ims_malignant, dtype='uint8')
```

Labelling is a very important part of the getting the data ready for training. CNN is a supervised learning algorithm and therefore needs to know what it is learning and eventually learn that benchmark features before starting to test. Another thing above is that we have created splits of the datasets and now let us merge them and shuffle them up with random.shuffle function.

```
y_benign = np.zeros(X_benign.shape[0])
y_malignant = np.ones(X_malignant.shape[0])

y_benign_test = np.zeros(X_benign_test.shape[0])
y_malignant_test = np.ones(X_malignant_test.shape[0])
```

```
X_train = np.concatenate((X_benign, X_malignant), axis = 0)
y_train = np.concatenate((y_benign, y_malignant), axis = 0)

X_test = np.concatenate((X_benign_test, X_malignant_test), axis = 0)
y_test = np.concatenate((y_benign_test, y_malignant_test), axis = 0)



s = np.arange(X_train.shape[0])
np.random.shuffle(s)
X_train = X_train[s]
y_train = y_train[s]

s = np.arange(X_test.shape[0])
np.random.shuffle(s)
X_test = X_test[s]
y_test = y_test[s]
```

Machine learning cannot understand any of the categorical data and we will need to encode our Melanoma and Benign into 0 and 1, this step is popularly called as Hot Encoding. We will also need to normalize the data by dividing the RGB values of the pixels by 255.

We achieve both of those steps by,

```
y_train = to_categorical(y_train, num_classes= 2)
y_test = to_categorical(y_test, num_classes= 2)
```

```
X_train = X_train/255.
X_test = X_test/255.
```

Before we start training our model, we will want to have 2 different version to explain. Our first model would be the standard CNN model and our next model is ResNet50. ResNet50 is 50 deep layer CNN, predominantly good to processed data which is apt for our case here. The complexity of the problem and image dataset imposing so many varying features, CNN cannot simply provide satisfying results and to have a better result, we are recommending a ResNet50 part of Keras. With ResNet50 we can increase the efficiency with keeping the training loss at a minimum. But the catch is that the training of this model takes huge time,

Fun facts:

Our algorithm (with Epoch value of 50) took 6693 second for one epoch and whopping 52 hours of continuous GPU computation. Given our limited resources and time span, we decided to shift the epoch value to 10. We might comprise on the overall accuracy but it is still much better performance than the CNN.

Model #1: CNN

```
def build(input_shape= (224,224,3), lr = 1e-3, num_classes= 2,
          init= 'normal', activ= 'relu', optim= 'adam'):
    model = Sequential()
    model.add(Conv2D(64, kernel_size=(3, 3),padding = 'Same',input_shape=input_sh
ape,
                     activation= activ, kernel_initializer='glorot_uniform'))
    model.add(MaxPool2D(pool_size = (2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(64, kernel_size=(3, 3),padding = 'Same',
                     activation =activ, kernel_initializer = 'glorot_uniform'))
    model.add(MaxPool2D(pool_size = (2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_initializer=init))
    model.add(Dense(num_classes, activation='softmax'))
    model.summary()

    if optim == 'rmsprop':
        optimizer = RMSprop(lr=lr)

    else:
        optimizer = Adam(lr=lr)

    model.compile(optimizer = optimizer ,loss = "binary_crossentropy", metrics=["
accuracy"])
    return model
```

https://keras.io/api/

This above link gives you almost everything we have used in both the models. The sequential API is used here. Model.sequential in the code triggers it and by add we are going to add layers. As the first add function is Conv2D layer with 64 filters. The second layer is also an exact copy of the first layer. The pooling (MaxPool2D) layer is the second most essential layer in CNN. This layer is basically a downsampling filter. It takes the maximum value from the two adjoining pixels. These are used to decrease computing cost and, to a lesser extent, overfitting. We must select the pooling size the greater the pooling dimension, the greater the importance of downsampling.

Other important points to explain in this part:

- Our activation function is relu (rectifier)
- Dropout is a regularization method. https://keras.io/api/layers/regularization_layers/dropout/
- Flatten layer is another function of the Keras to convert the feature map into a single way vector, as the name speaks for itself.
- We have used Adam Optimizer.

```
lr_reduction = ReduceLROnPlateau(monitor='val_acc',
                                            patience=5,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=1e-7)
```

https://keras.io/api/callbacks/reduce_lr_on_plateau/

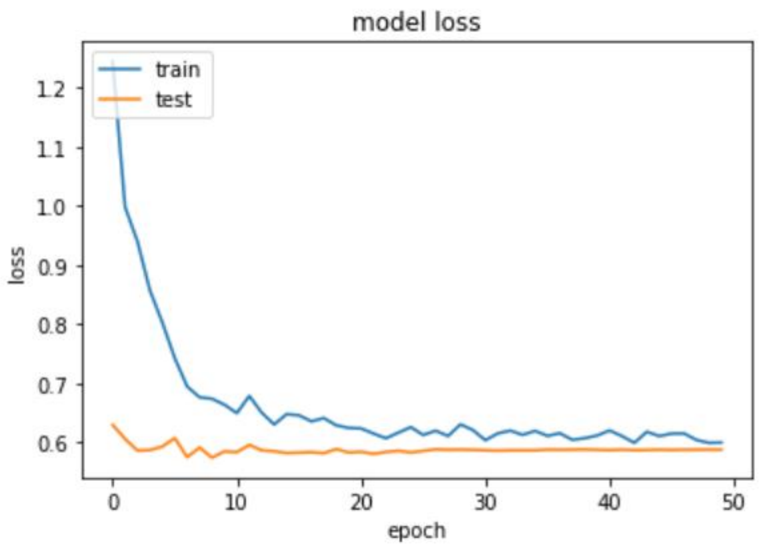
When the metric has stopped improving, we will reduce the learning rate with this function.

Let us try to train the model now,

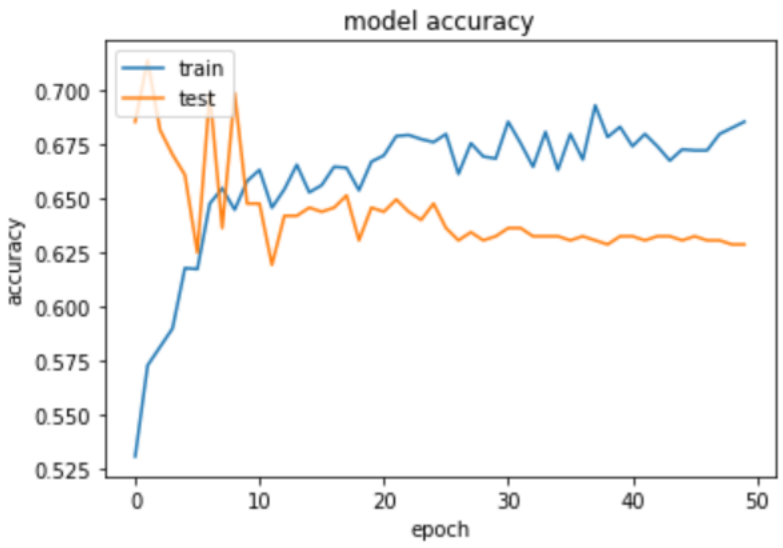
```
input_shape = (224,224,3)
lr = 1e-5
init = 'normal'
activ = 'relu'
optim = 'adam'
epochs = 50
batch_size = 64

model = build(lr=lr, init= init, activ= activ, optim=optim, input_shape= input_sh
ape)

history = model.fit(X_train, y_train, validation_split=0.2,
                    epochs= epochs, batch_size= batch_size, verbose=0,
                    callbacks=[lr_reduction]
                   )
```

```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

After the models have trained, the model accuracy and model loss graphs are:



The



The final step in this section is to test our CNN model:

```
model = build(lr=lr,
              init= init,
              activ= activ,
              optim=optim,
              input_shape= input_shape)

model.fit(X_train, y_train,
          epochs=epochs, batch_size= batch_size, verbose=0,
          callbacks=[learning_rate_reduction]
         )
```

```
y_pred = model.predict_classes(X_test)

Acc = accuracy_score(np.argmax(y_test, axis=1),y_pred))
Print("Test accuracy: ", Acc**100)
```

```
Test Accuracy: 65%
```

Now that we have CNN accuracy at 65%. We can start with our ResNet50 model.

To add more information about this,

https://keras.io/api/applications/resnet/#resnet50-function

We are going to reuse the same lr_reduction as we did in the CNN model for the ResNet50 model. We can start training the model now,

```
Before training remember to delete all the previous mod
els. input_shape = (224,224,3)
lr = 1e-5
epochs = 50
batch_size = 64

model = ResNet50(include_top=True,
                 weights= None,
                 input_tensor=None,
                 input_shape=input_shape,
                 pooling='avg',
                 classes=2)

model.compile(optimizer = Adam(lr) ,
              loss = "binary_crossentropy",
              metrics=["accuracy"])

history = model.fit(X_train, y_train, validation_split=0.2,
                    epochs= epochs, batch_size= batch_size, verbose=2,
                    callbacks=[lr_reduction]
                   )

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

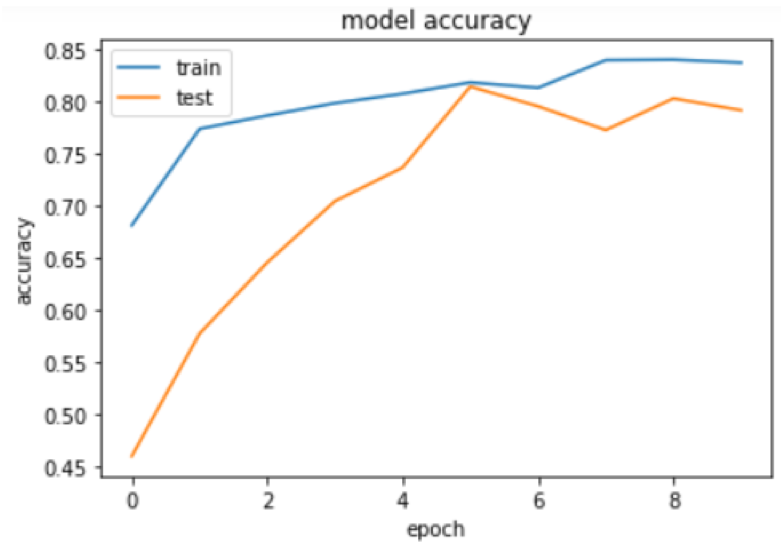
Understanding the arguments for ResNet50 model,

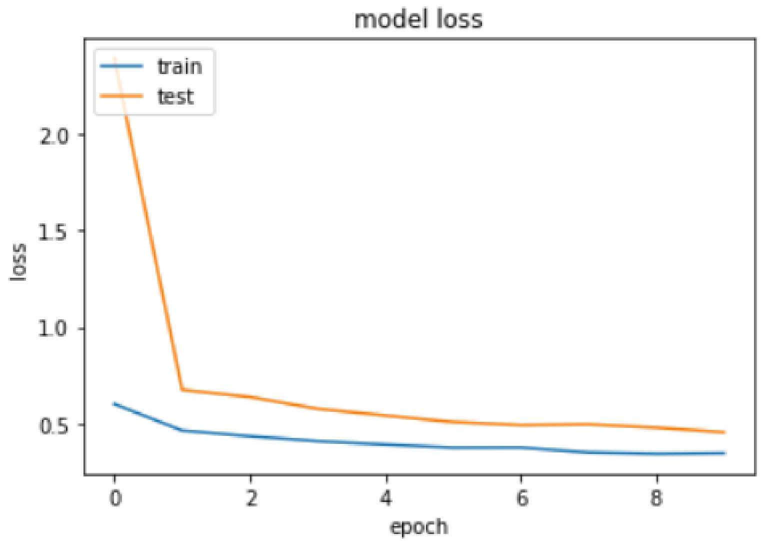
- **include_top**: whether to include the fully-connected layer at the top of the network.
- **weights**: one of None (random initialization), 'imagenet' (pre-training on ImageNet), or the path to the weights file to be loaded.
- **input_tensor**: optional Keras tensor (i.e. output of layers.Input()) to use as image input for the model.
- **input_shape**: optional shape tuple, only to be specified if include_top is False (otherwise the input shape has to be (224, 224, 3) (with 'channels_last' data format) or (3, 224, 224) (with 'channels_first' data format). It should have exactly 3 inputs channels, and width and height should be no smaller than 32. E.g. (200, 200, 3) would be one valid value.
- **pooling**: Optional pooling mode for feature extraction when include_top is False.
    - None means that the output of the model will be the 4D tensor output of the last convolutional block.
    - avg means that global average pooling will be applied to the output of the last convolutional block, and thus the output of the model will be a 2D tensor.
    - max means that global max pooling will be applied.
- **classes**: optional number of classes to classify images into, only to be specified if include_top is True, and if no weights argument is specified.

We have reproduced the same code from the CNN part in the plot functions. The next step is to fit the model and then test the model. Also save the model as well. We can share the result as well via this.

```
model.fit(X_train, y_train,
          epochs=epochs, batch_size= epochs, verbose=0,
          callbacks=[lr_reduction]
         )
y_pred = model.predict(X_test)
print(accuracy_score(np.argmax(y_test, axis=1), np.argmax(y_pred, axis=1)))
resnet50_json = model.to_json()

with open("resnet50.json", "w") as json_file:
    json_file.write(resnet50_json)

model.save_weights("resnet50.h5")
print("Saved model to disk")
```

```
Train on 2109 samples, validate on 528 samples
Epoch 1/10
 - 1707s - loss: 0.6011 - acc: 0.6814 - val_loss: 2.3934 - val_acc: 0.4602
Epoch 2/10
 - 1768s - loss: 0.4634 - acc: 0.7738 - val_loss: 0.6744 - val_acc: 0.5777
Epoch 3/10
 - 1780s - loss: 0.4342 - acc: 0.7866 - val_loss: 0.6375 - val_acc: 0.6458
Epoch 4/10
 - 1780s - loss: 0.4084 - acc: 0.7985 - val_loss: 0.5763 - val_acc: 0.7045
Epoch 5/10
 - 1789s - loss: 0.3909 - acc: 0.8075 - val_loss: 0.5415 - val_acc: 0.7367
Epoch 6/10
 - 1771s - loss: 0.3747 - acc: 0.8184 - val_loss: 0.5076 - val_acc: 0.8144
Epoch 7/10
 - 1886s - loss: 0.3754 - acc: 0.8132 - val_loss: 0.4915 - val_acc: 0.7955
Epoch 8/10
 - 2852s - loss: 0.3504 - acc: 0.8397 - val_loss: 0.4955 - val_acc: 0.7727
Epoch 9/10
 - 1841s - loss: 0.3433 - acc: 0.8402 - val_loss: 0.4788 - val_acc: 0.8030
Epoch 10/10
 - 2809s - loss: 0.3465 - acc: 0.8374 - val_loss: 0.4545 - val_acc: 0.7917
dict_keys(['val_loss', 'val_acc', 'loss', 'acc', 'lr'])
```





```
0.8287878787878787
Saved model to disk
```

The model has provided 82% efficiency given the reduced epoch size in our case. If the same algorithm is executed for a long time on this system or better resource then the accuracy **might** increase

Paper References and Important Links

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8160886/

https://www.mathworks.com/help/deeplearning/ref/resnet50.html;jsessionid=033c0c8b56451729b9204a5714bd

SVM and Snake Model: https://ieeexplore.ieee.org/document/8369708

KNN and SVM: **Detection** of **skin cancer using** SVM, random forest and **kNN** classifiers