Machine Learning Engineer Nanodegree

Capstone Project

Histopathologic Cancer Detection

Identify metastatic tissue in histopathologic scans of lymph node sections using deep learning

Lamess Hashad
September 10th, 2019

I. Definition

Project Overview

Many cancer deaths are caused when cancer moves from the original tumor and spreads to other tissues and organs. This is called metastatic cancer. In metastasis, cancer cells break away from the original (primary) tumor, travel through the blood or lymph system, and form a new tumor in other organs or tissues of the body. lymph node metastases occur in most cancer types (e.g. breast, prostate, colon). Lymph nodes are small glands that filter lymph, the fluid that circulates through the lymphatic system.

Histopathology refers to the microscopic examination of a biopsy or surgical specimen by a pathologist, after the specimen has been processed and histological sections have been placed onto glass slides.

Histologic image assessment has remained experience-based qualitative, and it always causes intra- or inter-observers variation even for experienced pathologists. This ultimately could result in inaccurate diagnosis which could lead to severe overtreatment or undertreatment, thus causing serious harms to patients. While prognosis is poorer when cancer has spread to the lymph nodes, the diagnostic procedure for pathologists is, however, tedious and time-consuming and prone to misinterpretation. Automated detection of lymph node metastasis has a great potential to help the pathologist and reduce their workload. Digital pathology, which is the digitization of glass slides at high resolution using whole-slide scanners, makes computerized quantitative analysis of histopathology imagery possible. Within the past few years, the field has been moving towards grand goals with strong potential diagnostic impact: (fully) automated analysis of whole-slide images to detect or grade cancer, to predict prognosis or identify metastases.

Recently, Deep Learning architectures, have shown its advantageous in image analysis over other non-deep learning based approaches. In histological image analysis community deep learning based approaches have evoked great interests since the pioneer work in [Ciresan DC et al (2013) Mitosis detection in breast cancer histology images with deep neural networks. In: MICCAI 2013. LNCS, vol 8150. Springer, Berlin, pp 411–418] [1] [2] [3] [4]

Dataset used:

I am using the dataset provided by Kaggle here. It contains a large number of small pathology images to classify. Files are named with an image id, data contains:

- train folder: contains 220,025 tiff image files (96 x 96px)
- train_labels.csv file: provides the ground truth for the images in the train folder.
- test folder: contains 57,458 tiff image files (96 x 96px) (this folder is not used in this project)

I am going to only use the images from the train folder (220025 images) as a subset of the data. This subset is going to be split into training\test\validation subsets (0.6\0.2\0.2).

A positive label indicates that the center 32x32px region of a patch contains at least one pixel of tumor tissue. Tumor tissue in the outer region of the patch does not influence the label. This outer region is provided to enable fully-convolutional models that do not use zero-padding, to ensure consistent behavior when applied to a whole-slide image.

This data set is a slightly modified version of the PatchCamelyon (PCam) benchmark dataset (the original PCam dataset contains duplicate images due to its probabilistic sampling, however, the version presented on Kaggle does not contain duplicates). PCam is derived from the Camelyon16 Challenge, which contains a total of 400 whole-slide images (WSIs) of sentinel lymph node from two independent datasets collected in Radboud University Medical Center (Nijmegen, the Netherlands), and the University Medical Center Utrecht (Utrecht, the Netherlands).

PCam is highly interesting, according to the author “It packs the clinically-relevant task of metastasis detection into a straight-forward binary image classification task, akin to CIFAR-10 and MNIST. Models can easily be trained on a single GPU in a couple hours, and achieve competitive scores in the Camelyon16 tasks of tumor detection and WSI diagnosis. Furthermore, the balance between task-difficulty and tractability makes it a prime suspect for fundamental machine learning research on topics as active learning, model uncertainty and explainability.” [5] [6] [4]

Problem Statement

This project is based on the Kaggle competition: Histopathologic Cancer Detection. The objective is using deep learning to classify small image patches (96 x 96px) taken from larger digital pathology scans to images that contain metastatic cancer or not (at the center 32x32px region).

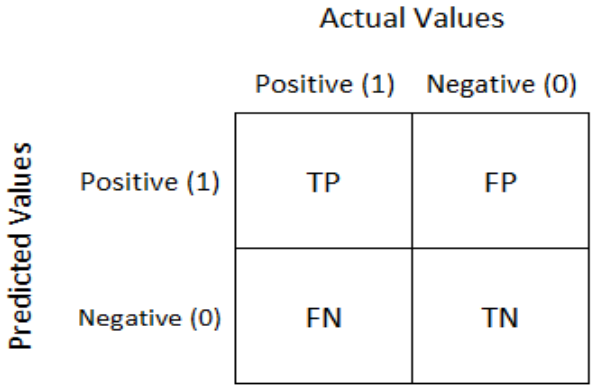
To achieve the solution I’m using transfer learning with convolutional neural networks to build a deep learning classifier model and train it on the training data. I based my solution on this paper [Nguyen, Long & Lin, Dongyun & Lin, Zhiping & Cao, Jiuwen. (2018). Deep CNNs for microscopic image classification by exploiting transfer learning and feature concatenation.] [5] [6] [7]

Metrics

Evaluation metrics used for this project are:

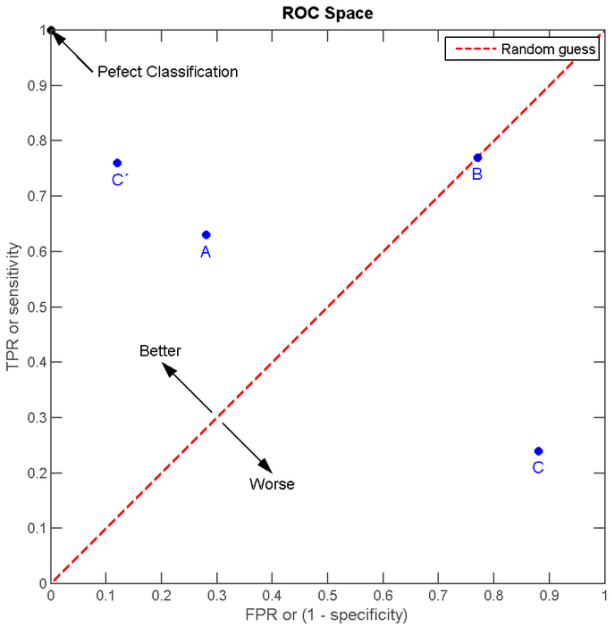
- Confusion matrix (true positive/negative, false positive/negative), based on its values we can calculate a lot of other evaluation metrics. In general, the more ‘accurate’ a system, the greater number of true positives and true negatives occur, and the fewer false positives and

false negatives. I am going to focus on the false negative value and try to make it as low as possible, and that is because in medical applications like this one false negatives are the most dangerous, as it means sending patients home while they need treatment.



Confusion Matrix

- Area under the ROC curve (AUC), this is the evaluation used by Kaggle to evaluate the original challenge, also it's one of the evaluation metrics reported in the benchmark I'm using. The higher the curve towards the upper left corner, the more accurate the test is.



ROC curve

- Accuracy, number of correct predictions made divided by the total number of predictions made. It's also one of the evaluation metrics reported in the benchmark I'm using.
- Diagnostic odds ratios(DOR), it's a measurement of the diagnostic accuracy of a test, it combines positive and negative likelihood ratios incorporating all four values of the confusion matrix. The advantage of DOR is that it's medically recognized.

$$DOR = LR+ /LR- = (TPR/FPR) / (FNR/TNR)$$

$$= (\text{sensitivity/false positive rate}) / (\text{false negative rate/specificity})$$

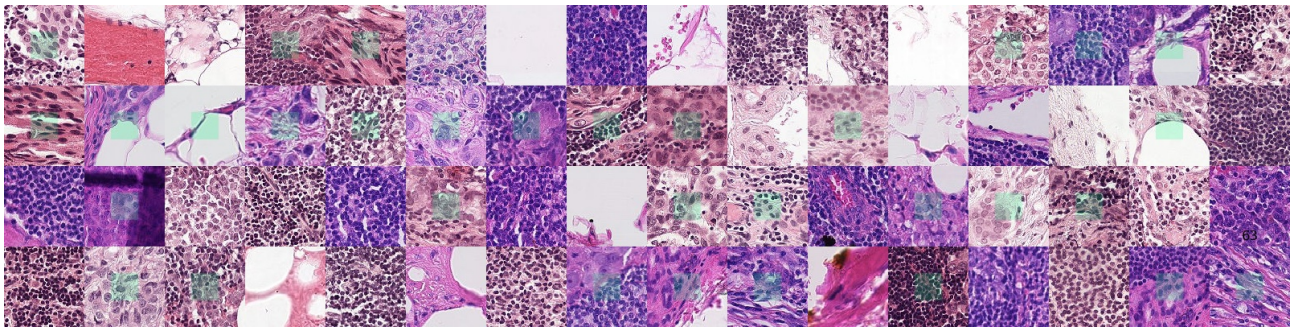
[5] [6] [8]

II. Analysis

Data Exploration and Visualization

As mentioned before the dataset I'm using is a subset of the Kaggle dataset Histopathologic Cancer Detection. This subset is the images from the dataset's train folder in addition to the csv file containing the labels for these images.

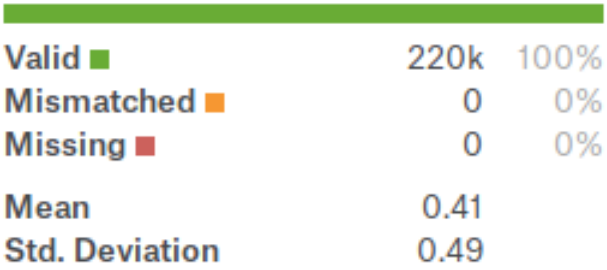
The dataset contains 220,025 tif colored (3 channels) image files (96 x 96px) extracted from whole-slide images (WSIs) histopathologic scans of lymph node sections, each image file is named with an id. The csv file contains two columns (id, label), a positive label indicates that the center 32x32px region of a patch contains at least one pixel of tumor tissue, a negative label indicates that no tumor was found. [5] [6]



Example images from PCam. Green boxes indicate tumor tissue in center region, which dictates a positive label. [6]

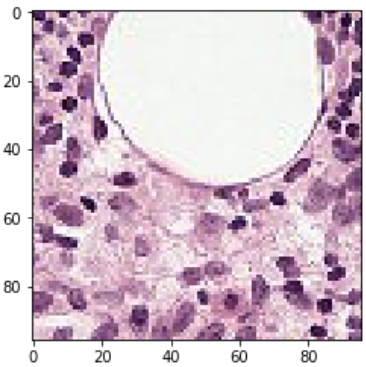
Of the 220,025 images, 130,908 are labeled negative, and 89.117 are labeled positive. []

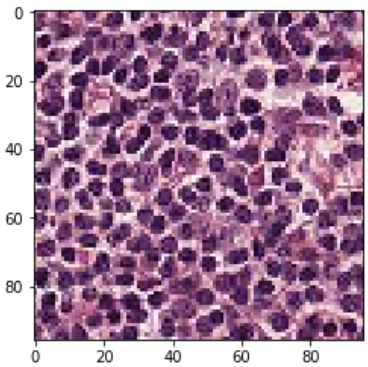
[5]

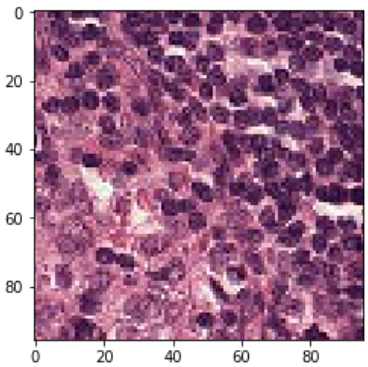


[5]

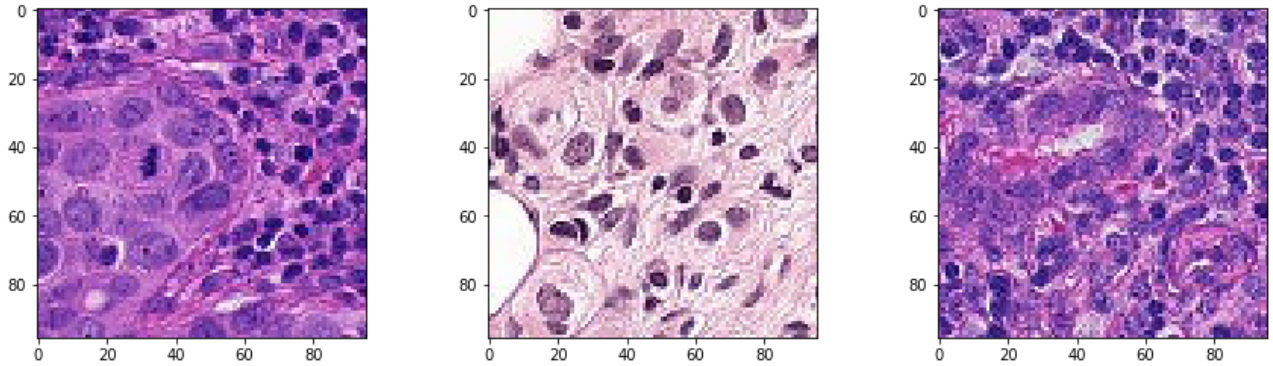
Examples of negative labeled images:







Examples of positive labeled images:



Algorithms and Techniques

As mentioned before, my solution is based on this paper [7]. The idea is to use transfer learning to build a classifier by concatenating the features extracted from three different pretrained deep CNNs to improve classification accuracy. The concatenated features are then used to train two fully-connected layers to adopt these generic features to biomedical data and perform classification. According to the paper this technique shows significant performance gains on the The PAP-smear and 2D-Hela datasets (microscopic medical datasets), compared to the neural network structure that uses only features extracted from single CNN and several traditional classification methods.

Methods	*2D-Hela*	*PAP smear*
Single transfer learning network with Inception-v3 [23]	90.72 ± 1.85	89.66 ± 1.89
Single transfer learning network with Resnet152 [23]	89.72 ± 2.18	90.87 ± 1.48
Single transfer learning network with Inception – Resnet v2	92.00 ± 1.97	89.25 ± 2.23
Proposed features concatenation network	**92.57 ± 2.46**	**92.63 ± 1.68**

The format of the table is accuracy ± std (%)

classification accuracy comparison with transfer learning methods [7]

Methods	*2D-Hela*	*PAP smear*
SIFT(BoW(VQ)+SPM+SVM) [24]	83.79 ± 2.5	84.03 ± 2.3
LBP(BoW(VQ)+SPM+SVM) [25]	81.47 ± 2.1	81.43 ± 2.1
SAHLBP(BoW(VQ)+SPM+SVM) [3]	84.49 ± 2.2	86.21 ± 2.0
SIFT+SAHLBP(BoW(VQ)+SPM+SVM) [3]	86.20 ± 2.5	87.63 ± 2.1
SIFT(BoW(LLC)+SPM+Softmax) [6]	89.37 ± 1.5	89.96 ± 1.4
Proposed features concatenation network	**92.57 ± 2.46**	**92.63 ± 1.68**

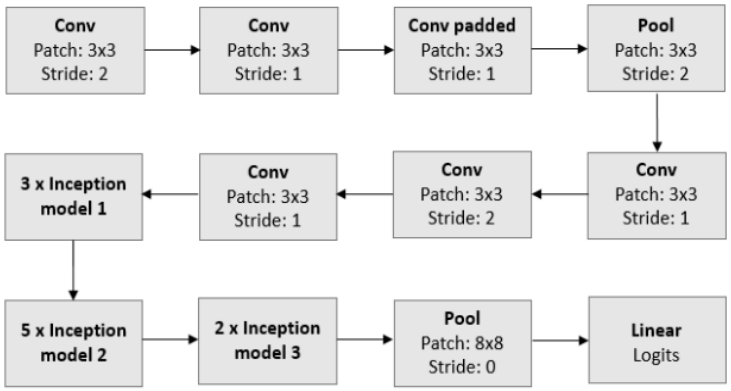
The format of the table is accuracy ± std (%)

classification accuracy comparison with existing methods [7]

For biomedical image classification the number of training images is often limited, so transfer learning using deep CNNs is often applied. Such deep CNNs while performing very well on large datasets (e.g. ImageNet) they may suffer from serious overfitting on smaller datasets (like biomedical image datasets), and that's why we use transfer learning with pretrained CNNs to avoid training the CNN from scratch hence avoid overfitting.

In this solution I'm using three deep CNN architectures: Inception-v3, Resnet152 and Inception-Resnet-v2 as the feature extractors (top layers are not included), these CNNs are pretrained on a nature image dataset (ImageNet: over 1 million training images) and they're capable to learn generic image features that are applicable to other image datasets without training from scratch.

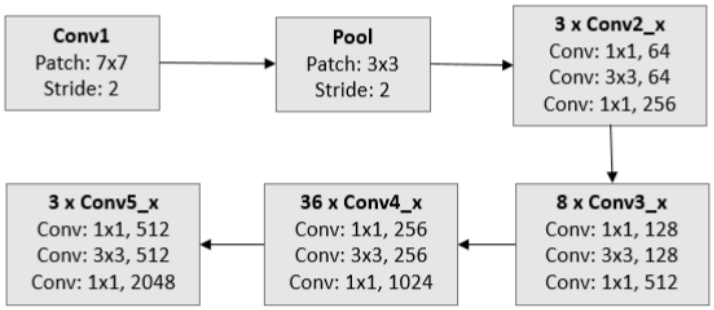
The details of the features generated by the pretrained deep CNNs (after taking our dataset as input) are summarized as follows:



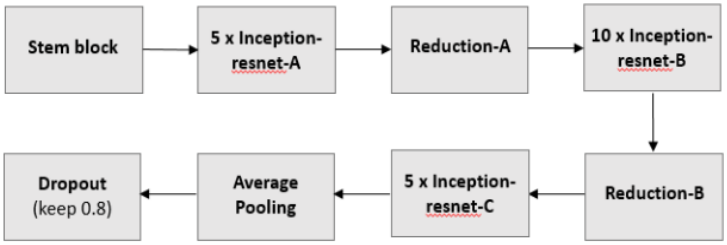
Inception-v3: For one image, we extract a 2048-dimensional feature from the last fully-connected logits layer.

The basic architecture of Inception-v3 [7]

Resnet152: For one image, we extract a 2048-dimensional feature from the last fully-pooling layer (Conv5x layer).



The basic architecture of Resnet152 [7]



Inception-Resnet-v2: For one image, we extract a 1536-dimensional feature from the last fully connected layer after dropout.

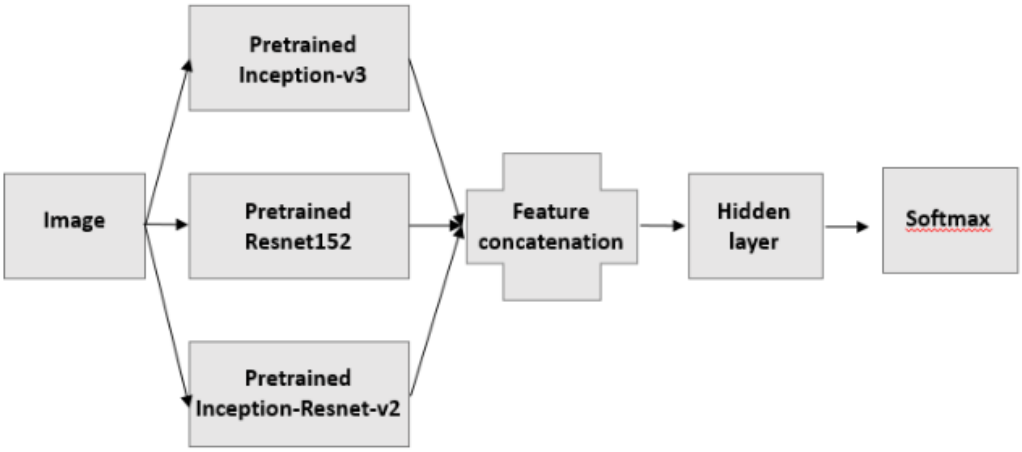
The basic architecture of Inception-Resnet-v2 [7]

After that we concatenate the extracted features from three CNNs to form a 5632 dimensional feature vector, since different CNN architectures can capture diverse information in microscopic

images, such concatenation of multiple CNN features integrates the information from different CNNs together to create a more discriminative feature representation compared with a single CNN structure. Last, we feed the concatenated feature vector into two fully-connected layers for classification.

Parameters:

- units for the first Dense layer is 512, and for the final Dense layer is 1.
- activation: for the hidden layer ‘relu’, and for the final layer sigmoid
- weights initializer in the fully-connected layers: ‘RandomNormal(mean=0, stddev=0.001)’
- model optimizer: ‘RMSprop’
- learning rate and decay: I’m going to try different values until I get the best performance.
- number of epochs: I’m going to start with 30 and maybe decrease it if too much.



Proposed feature concatenation network structure [7]

Other techniques I’m intending to use:

- image augmentation: because the data is limited, I’m going to use image augmentation to artificially create more training images from the ones I have through different ways of processing, such as random rotation, width and height shifts, zoom and vertical and horizontal flips.
- early stopping: to stop the training before the model starts overfitting, and restore the best weights. [7]

Benchmark

As a benchmark I will use the results reported under the benchmark section in the README.md file of *The PatchCamelyon (PCam) deep learning classification benchmark* github repository (accuracy: 89.8%, area under the ROC curve: 96.3) [6]

III. Methodology

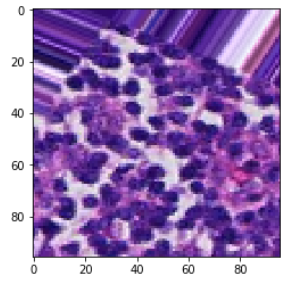
Data Preprocessing

Preprocessing steps:

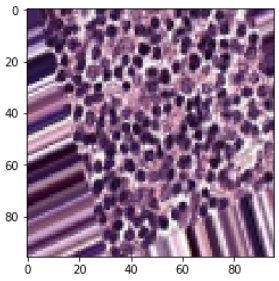
- Split the data into train\validation\test (0.6\0.2\0.2) subsets, then split each subset into two sub-folders representing the two classes of the data, the classes sub-folders are necessary for the flow_from_directory method.
- Normalization: Neural networks process inputs using small weight values, and inputs with large integer values can disrupt or slow down the learning process. As such it is good practice to normalize the pixel values so that each pixel value has a value between 0 and 1. This can be achieved by dividing all pixels values by the largest pixel value; that is 255. This is performed across all channels, regardless of the actual range of pixel values that are present in the image.
- Image augmentation: One of the best ways to improve the performance of a Deep Learning model is to add more data to the training set. Data augmentation is a strategy that enables us to significantly increase the diversity of data available for training models, without actually collecting new data. As mentioned before this dataset is relatively limited so I account for this by training the model on additional synthetically modified data. Augmentations used are (rotation, width and height shift, zoom, horizontal and vertical flip)

First, using the os library, I split the data into train and test subsets (0.8\0.2), each split into two classes (sub-folders 0 and 1). Then using ImageDataGenerator class from Keras, I create a generator that loops over the data generating batches of tensor image data with real-time data augmentation, also I use it to split the training data into train\validation (0.75\0.25) subsets using the argument (validation_split = 0.25), also I use it to normalize all the images using the argument (rescale = 1/255) to transform every pixel value from range [0,255] -> [0,1]. After that I use the method (flow_from_directory) to apply the generator to the data. Augmentation is applied only on the training data and validation data but not on the test data. [9] [10] [11] [12] [13]

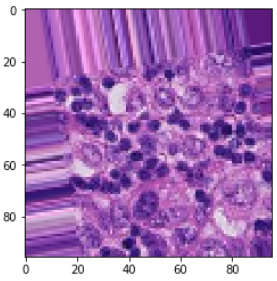
Example of images after preprocessing:



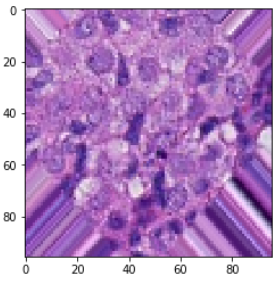
class 0



class 0



class 1



class 1

Implementation

This project was implemented using python 3 in a google colaboratory notebook with GPU as a hardware accelerator, the deep learning library Keras with TensorFlow as a backend and Kaggle API to download the dataset.

Implementation steps:

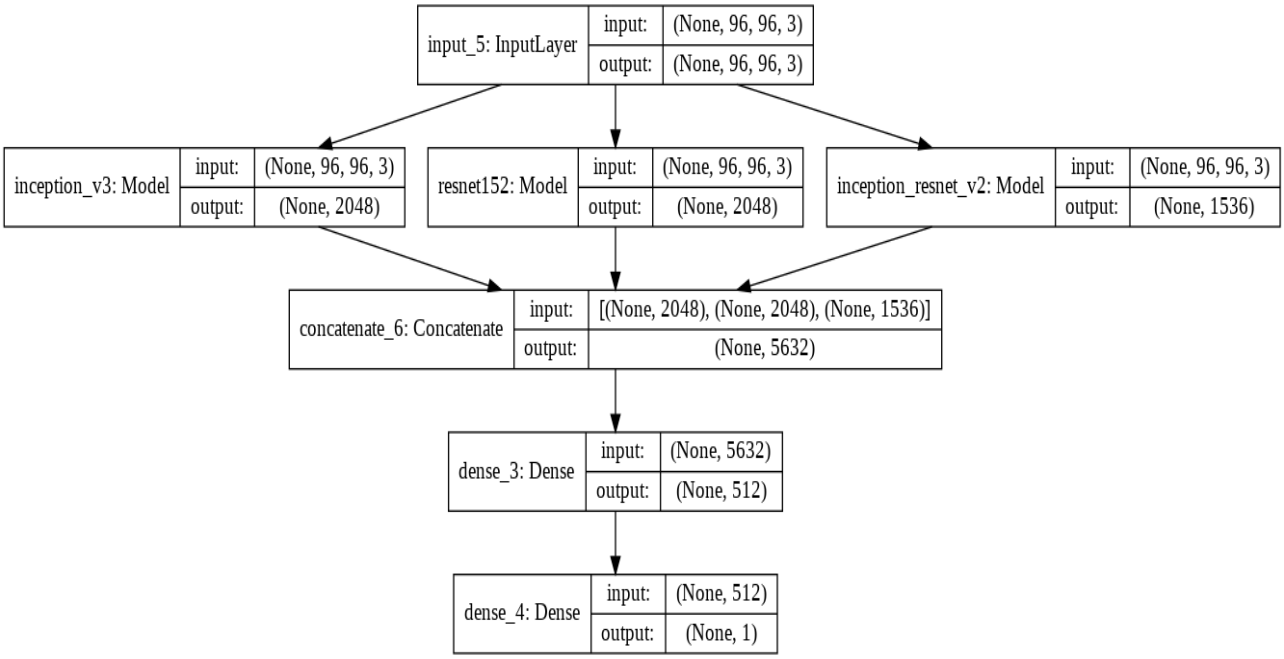
1. import all the necessary libraries\functions: os, numpy, Keras(ImageDataGenerator, Input, Dense, InceptionV3, ResNet152, InceptionResNetV2, RandomNormal, Model, plot_model, ModelCheckpoint, EarlyStopping, RMSprop), random, matplotlib(plt, mpimg), sklearn(confusion_matrix, accuracy_score, roc_curve, auc)
2. create the necessary directories: "/capstone_project/", "/capstone_project/data/", "/capstone_project/saved_models/"
3. set up the kaggle API: install kaggle and upload kaggle.json
4. download the dataset from kaggle
5. unzip the data from train.zip and train_labels.csv
6. use operations from the os library to split the data into train\test (0.8\0.2) subsets, and each subset into two sub-folders for each class.
7. Show random images from the positive and negative classes
8. split the training subset into training\validation (0.75\0.25), normalize all images and apply augmentation using (ImageDataGenerator, flow_from_directory). Test images are normalized but not augmented. The batch_size that worked in my case was 100.
9. show random images from the three subsets (train, validation, test)
10. create the model: create Input layer with shape (96,96,3), then create inception_v3, resnet152 and inception_resnet_v2 models with the ‘imagenet’ weights, ‘avg’ pooling and top not included, after that Concatenate, after that add two fully connected layers with RandomNormal weights initializer (zero mean and 0.001 standard deviation), first Dense layer has 512 nodes and ‘relu’ activation function, the second Dense layer has 1 node and ‘sigmoid’ activation function.

    After that compile the model with RMSprop optimizer (learning rate = 0.0001 and decay = 0.001) and ‘binary_crossentropy’ loss function.

    Finally show the summary and plot the model
11. train the model: using fit_generator train the model for 10 epochs, use ModelCheckpoint and EarlyStopping callbacks to save the model after every epoch and stop training when validation accuracy stops improving respectively. EarlyStopping also helped me to stop the model when the accuracy was improving very slowly as the GPU time on google colaboratory is limited.
12. Plot training and validation accuracy, plot training validation loss
13. predict test images labels: using predict_generator, this outputs an array of probabilities for each image, so create another array with binary labels instead of probabilities.

14. Get true positive, true negative, false positive and false negative values and plot the confusion matrix using sklearn's confusion_matrix. I used a modified function from sklearn examples to plot the confusion matrix.
15. Plot the ROC curve and get the area under the curve using sklearn's roc_curve and auc
16. Get the accuracy using sklearn's accuracy_score.
17. Finally calculate Diagnostic odds ratios (DOR).

[14] [15] [16] [17] [9] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31]



model architecture

Refinement

My initial model was the same as my final model except for the learning rate and decay values, my initial model used 'rmsprop' optimizer with the default learning rate of 0.001 and decay of 0.0, my final model also uses rmsprop as optimizer but with learning rate of 0.0001 and decay of 0.001.

The initial model wasn't able to reach a good accuracy and the validation accuracy started to decline in the second epoch so it was stopped by the EarlyStopping callback. Removing the EarlyStopping callback, the model accuracy was improving each epoch reaching 0.94 in the sixth epoch, while the validation accuracy didn't reach 0.91 until the sixth epoch, which means that the model was learning very slowly or started to overfit.

```
Epoch 1/10
1321/1321 [==============================] - 4617s 3s/step - loss: 0.2803 - acc: 0.8860 - val_loss: 0.2775 - val_acc: 0.8990

Epoch 00001: val_acc improved from -inf to 0.89898, saving model to /capstone_project/saved_models/weights.01-0.90.hdf5
Epoch 2/10
1321/1321 [==============================] - 4511s 3s/step - loss: 0.2148 - acc: 0.9174 - val_loss: 0.5643 - val_acc: 0.8468

Epoch 00002: val_acc did not improve from 0.89898
```

initial model training results with EarlyStopping

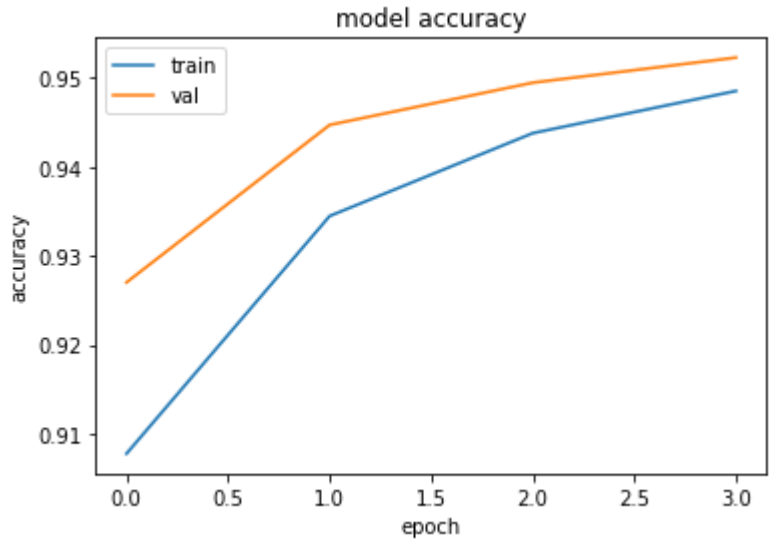
```
Epoch 1/10
1320/1320 [==============================] - 4615s 3s/step - loss: 0.2721 - acc: 0.8906 - val_loss: 0.2839 - val_acc: 0.8951

Epoch 00001: val_acc improved from -inf to 0.89510, saving model to /capstone_project/saved_models/weights.01-0.90.hdf5
Epoch 2/10
1320/1320 [==============================] - 4497s 3s/step - loss: 0.2094 - acc: 0.9194 - val_loss: 0.4019 - val_acc: 0.8757

Epoch 00002: val_acc did not improve from 0.89510
Epoch 3/10
1320/1320 [==============================] - 4499s 3s/step - loss: 0.1918 - acc: 0.9272 - val_loss: 0.8377 - val_acc: 0.7893

Epoch 00003: val_acc did not improve from 0.89510
Epoch 4/10
1320/1320 [==============================] - 4482s 3s/step - loss: 0.1797 - acc: 0.9334 - val_loss: 0.3623 - val_acc: 0.9054

Epoch 00004: val_acc improved from 0.89510 to 0.90540, saving model to /capstone_project/saved_models/weights.04-0.91.hdf5
Epoch 5/10
1320/1320 [==============================] - 4490s 3s/step - loss: 0.1689 - acc: 0.9377 - val_loss: 0.4192 - val_acc: 0.9083

Epoch 00005: val_acc improved from 0.90540 to 0.90831, saving model to /capstone_project/saved_models/weights.05-0.91.hdf5
Epoch 6/10
1320/1320 [==============================] - 4485s 3s/step - loss: 0.1630 - acc: 0.9409 - val_loss: 0.3037 - val_acc: 0.9078

Epoch 00006: val_acc did not improve from 0.90831
Epoch 7/10
1319/1320 [==============================>.] - ETA: 3s - loss: 0.1566 - acc: 0.9434
```

initial model training results without EarlyStopping

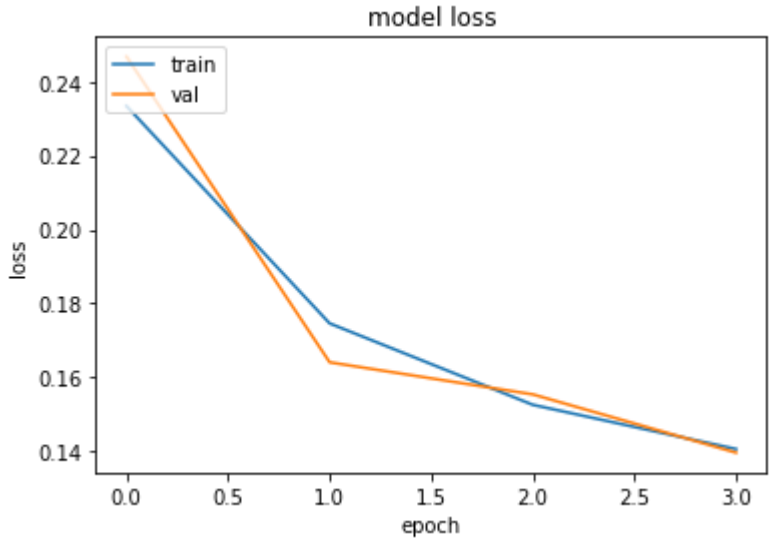
Trying to improve the model I decided to try reducing the learning rate and use a decay larger than zero. As mentioned my final model has a learning rate of 0.0001 and decay of 0.001, it was able to reach accuracy of 0.9486 and validation accuracy of 0.9523 after four epochs, than it was stopped by the EarlyStopping callback because it was improving very little.

```
Epoch 1/10
1322/1322 [==============================] - 1817s 1s/step - loss: 0.2333 - acc: 0.9078 - val_loss: 0.2467 - val_acc: 0.9270

Epoch 00001: val_acc improved from -inf to 0.92702, saving model to /capstone_project/saved_models/weights.01-0.93.hdf5
Epoch 2/10
1322/1322 [==============================] - 1709s 1s/step - loss: 0.1748 - acc: 0.9344 - val_loss: 0.1641 - val_acc: 0.9447

Epoch 00002: val_acc improved from 0.92702 to 0.94471, saving model to /capstone_project/saved_models/weights.02-0.94.hdf5
Epoch 3/10
1322/1322 [==============================] - 1708s 1s/step - loss: 0.1527 - acc: 0.9437 - val_loss: 0.1554 - val_acc: 0.9495

Epoch 00003: val_acc improved from 0.94471 to 0.94946, saving model to /capstone_project/saved_models/weights.03-0.95.hdf5
Epoch 4/10
1322/1322 [==============================] - 1707s 1s/step - loss: 0.1406 - acc: 0.9486 - val_loss: 0.1397 - val_acc: 0.9523

Epoch 00004: val_acc improved from 0.94946 to 0.95227, saving model to /capstone_project/saved_models/weights.04-0.95.hdf5
```

final model training results with EarlyStopping



final model accuracy plot



final model loss plot

IV. Results

Model Evaluation and Validation

As mentioned before my solution is based on this paper [7], it's a transfer learning network exploiting features concatenation from three deep CNNs, I only needed to tune optimizer, learning rate and decay to try to reach the best results I can. For a full description of the final model refer to number 10 of implementation steps in the methodology section of this report.

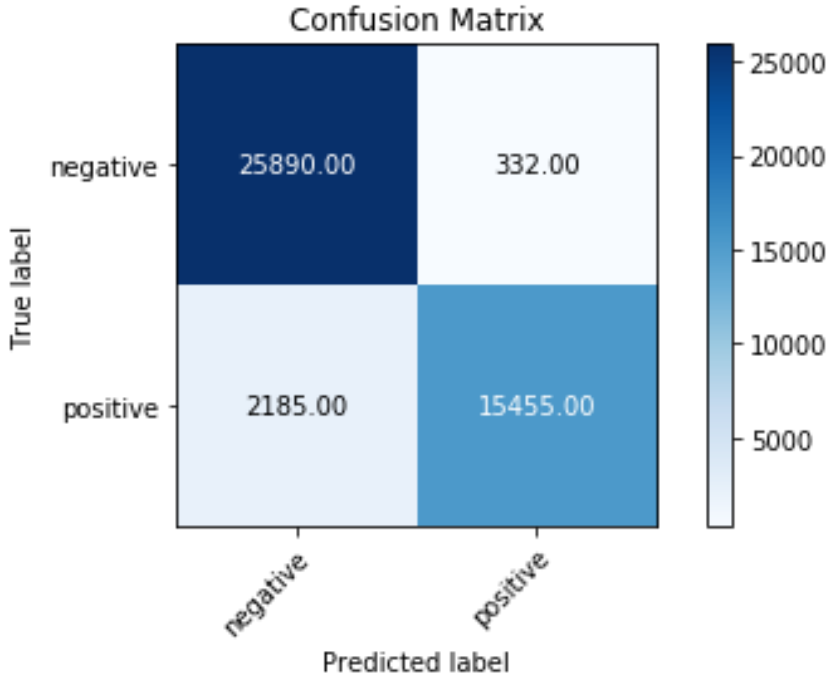
I chose this model because according to the results of the paper it performed very well on other medical microscopic datasets (2D-Hela, PAP smear), it shows superior performance to single CNN networks without concatenation and several traditional classification methods. Compared with the best competing method, the proposed method achieves significant accuracy gains (3.20% for 2D-Hela and 2.67% for PAP smear, respectively). Also since transfer learning is adopted, the proposed method produces good classification performance without training deep neural networks from scratch.

The test images were used to evaluate the performance of the model, these images are a separate subset of the data that was not used for training or validation. The validation accuracy during training reached 0.9520 and test accuracy was 09427, because the accuracy of the validation and testing are close we can conclude that the model is robust enough. [32]

Justification

Test results: (accuracy: 89.8%, area under the ROC curve: 96.3

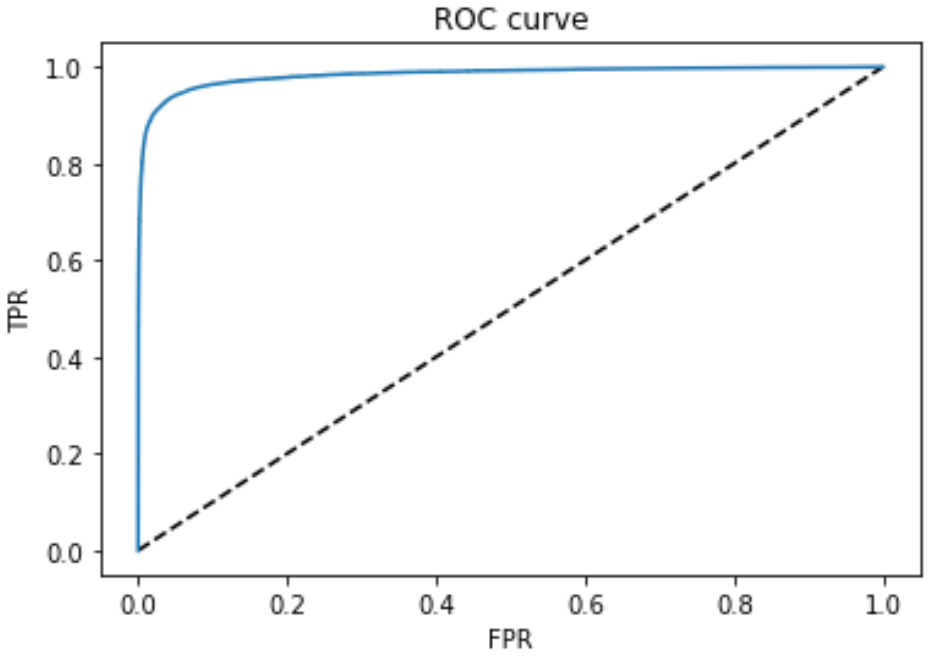
- Confusion Matrix:
  - true negative: 25890
  - false positive: 332
  - false negative: 2185
  - true positive: 15455



confusion matrix

- ROC curve and AUC:
  - AUC: 0.98381

  compared to the benchmark's AUC of 96.3%, this model is better by 2%



ROC curve

- Accuracy: 0.94262

  compared to the benchmark's accuracy of 89.8%, this model gained 4.5% of accuracy.
- Diagnostic odds ratios (DOR): 551.58384.

  This diagnostic odds ratio is greater than one, so we know that the test is discriminating correctly.

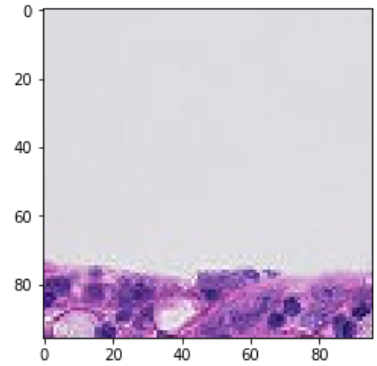
We can see from the results that the model successfully passed the benchmark with significant gain. [33]

V. Conclusion

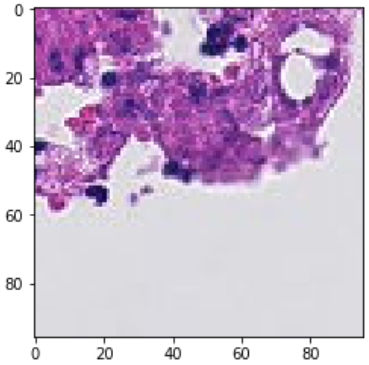
Free-Form Visualization

From the confusion matrix we can see that false negatives are significantly more than false positives, and as mentioned before, for medical applications it's important to focus on reducing false negatives. Visualizing examples of false negatives and false positives may help us understand the results and maybe help us find ways to improve the model. In my case I did this using the generator methods classes and filenames to find the true class of the image and get the file name respectively, using a for loop to loop through the true classes and predictions I was able to plot false negative and false positive images with some of the worst predictions. This can be done easier if using fastai.

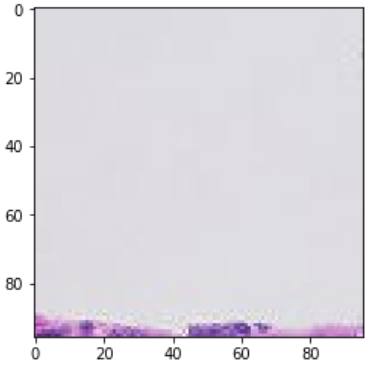
Examples of false negatives with very bad predictions (0):



prediction: 0

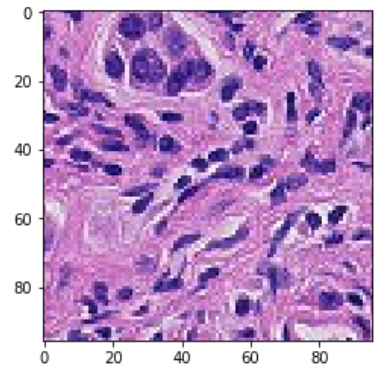


prediction: 0

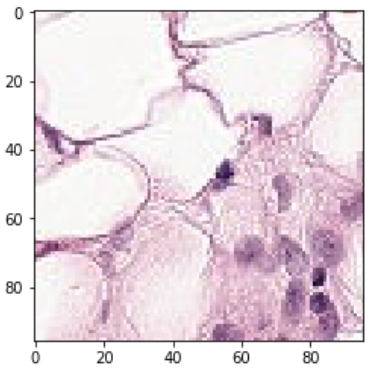


prediction: 0

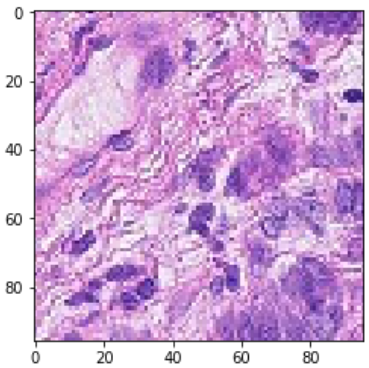
Examples of false positives with very bad predictions(very close to 1):



prediction: 0.99999166



prediction: 0.99997926



prediction: 0.99999505

Reflection

The process used for this project can be summarized in these steps:

1. Deciding on a domain and finding an interesting problem and a good dataset.
2. Downloading the dataset, split to train\validation\test subsets and applying preprocessing.
3. Building a transfer learning deep CNN model to classify the images.
4. Training the model

5. Test the model on test data and and compare the performance to the benchmark.
6. If results are not as expected, go back to step 3, refine the model and test again until reaching good results.

The first challenge I faced in this project was that my local hardware wasn't able to run the model, so I needed to familiarize myself with google colaboratory, this was easy with all the available tutorials. The second challenge was finding a model architecture suitable for the problem, reading papers and articles about projects dealing with similar datasets was a great help and as mentioned I finally decided to base my model on this paper [7]. The final challenge was refining the model to reach good results, I had to experiment with the learning rate and decay values until the model showed good results. My final model was able to pass the benchmark but I wasn't able to reduce the value of the false negatives below the reported value.

One of the interesting aspects of this project was dealing with microscopic medical data, learning the difficulties a deep learning model faces with such datasets (e.g: the data being limited and the training labels being vulnerable to human error) and learning how medical applications are evaluated and which metrics are the most important for such applications.

Improvement

Improvements that I think could be done:

- Refine the model to reduce the false negatives value, this could be done by analyzing the false negative images to find I they confuse the model
- one technique I wanted to use but didn't know how is One Cycle Learning Rate Policy, I think using this technique I can reach better results.
- When satisfied with the model results, we can try to modify the program to be able to deal with the whole-slide images from the original dataset CAMELYON16

---

sources:

[1] National Cancer Institute

[2] Wikipedia: histology page

[3] Xu J., Zhou C., Lang B., Liu Q. (2017) Deep Learning for Histopathological Image Analysis: Towards Computerized Diagnosis on Cancers. In: Lu L., Zheng Y., Carneiro G., Yang L. (eds) Deep Learning and Convolutional Neural Networks for Medical Image Computing. Advances in Computer Vision and Pattern Recognition. Springer, Cham

[4] Camelyon16 challenge site

[5] Histopathologic Cancer Detection: Identify metastatic tissue in histopathologic scans of lymph node sections, challenge page on Kaggle

[6] The PatchCamelyon (PCam) deep learning classification benchmark page on GitHub

[7] Nguyen, Long & Lin, Dongyun & Lin, Zhiping & Cao, Jiuwen. (2018). Deep CNNs for microscopic image classification by exploiting transfer learning and feature concatenation. 1-5. 10.1109/ISCAS.2018.8351550.

[8] Towards Data Science article:How data scientists can convince doctors that AI works

[9] Keras Documentation: Preprocessing » Image Preprocessing

[10] Machine Learning Mastery article: How to Manually Scale Image Pixel Data for Deep Learning

[11] Towards Data Science article: Data Augmentation on Images

[12] https://bair.berkeley.edu/blog/2019/06/07/data_aug/

[13] nanonets blog: Data Augmentation | How to use Deep Learning when you have Limited Data Part 2

[14] Keras Documentation: Applications

[15] Tutorial for Kaggle competition using Google Colab

[16] stack over flow: How to move a file in Python

[17] stack over flow: Split data directory into training and test directory with sub directory structure preserved

[18] Display an image with Python

[19] Keras Documentation: Layers » Core Layers

[20] Keras Documentation: Backend

[21] Keras Documentation: Initializers

[22] Keras Documentation: Getting started » Guide to the Sequential model

[23] Keras Documentation: Visualization

[24] Keras Documentation: Models » Model (functional API)

[25] Keras Documentation: Callbacks

[26] sklearn.metrics.confusion_matrix

[27] sklearn example: Confusion matrix

[28] sklearn.metrics.roc_curve

[29] sklearn example: Feature transformations with ensembles of trees

[30] sklearn.metrics.auc

[31] sklearn.metrics.accuracy_score

[32] ResearchGate: What is the definition of the robustness of a machine learning algorithm?

[33] wikipedia: Diagnostic odds ratio