Politecnico Di Torino

A.A. 2017-18

Corso di laurea magistrale in ingegneria Biomedica/ Informatica

Bioinformatics

TREE CNN FOR COLORECTAL CARCINOMA DIAGNOSIS

Francesco Alotto s242459

Riccardo Di Dio s235764

Ylenia Placanica s243838

INDEX:

DESCRIPTION OF THE PROJECT

The project is focused on the digital pathology field. This sphere of research points at developing tools usable by pathologists.
The aim of the project is to study how a Tree-CNN may grow in relation to the hierarchical structure of the dataset and to compare it with a classic 5 classe CNN.
Our Dataset composed by __ images divided into Healthy tissue (H) or Adenocarcinoma (cancer, AC) at the first level. At the second level, we keep having Healthy tissue and Adenoma (AD), a class that is a precursory lesion the may turn into cancer but is still non-cancerous at the current state, and may be split into three sub-classes of adenomas type: Serrated, Tubular or Villous (S,T,V). The dataset images are very complex and the classes are very similar one to each other. To exceed this problem, we tried to develop a Tree CNN as reported in the article [1]. Starting from the main root that divided Healthy cell from Cancer ones, we developed the rest of the Tree.

Our focus was the implementation of an incremental learning strategy, based on the network that have given the best result based on a 5-classes tree. So we tried to train different types of networks (vgg16, inceptionV3, densenet201, xception, mobileNet5) and noticed that the best results were given by the VGG16, so this one was the network choiced and we kept development with it. FIrst of all, after networks comparison discussing before, we focused on the accessibility of our framework. We didn't want to have a very powerful and precise restricted to only devices with an adequate hardware power. So we decided to use the concept of "microservice" which has made possible to build a distributed architecture, with a webapp and a Telegram Bot.

In the final part of the discussion, we have compared the Tree CNN Model with the 5 classes network, in order to objectively decide which one is the best.

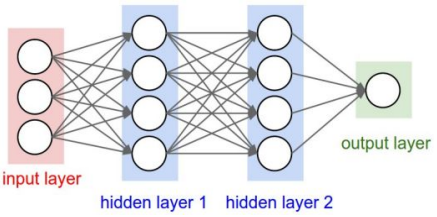
CONVOLUTIONAL NEURAL NETWORKS

Machine learning is a subfield of artificial intelligence that gives to computers the ability to learn how to solve a specific problem without being explicitly programmed for it. "Learning" is the ability to derive knowledge from experimental data.

Artificial Neural Networks (ANN or NN, Neural Networks) are computing systems inspired on the biological neural networks in animals brains. They are based on different algorithms that work together to process complex inputs. Those algorithms have completely revolutionized machine learning.

Differently from standard programming where from an input, coders develop an algorithm to get the output, in ML you provide to the net both input and output and the net builds the function. The training phase consists in feeding the net with many samples in order to modify the weights of each neuron in order to get the highest accuracy possible.

Convolutional Neural Networks (CNN) is a feed forward artificial neural network: connection patterns between neurons are not circular, but based on the shape of animal visual cortex, so the output of each neuron of each layer feeds all the neurons of the next layers.



CNN is composed of four main layers:

- **convolutional layer:** applies a convolution operation to data. This operation preserves features useful to classify images into classes. There can be more than 1 convolutional layer.
- **pooling layer:** combines the results of all the convolutional layers into a single one. We can use different types of pooling (max, min, average) depending on the type of generalization used to choose features.
- **flattening layer:** transforms the matrix of our results into a vector.
- **fully connected layer:** connect every neuron in one layer to every neuron in another layer.

Beyond these layers, we can also have a dropout layer used as a regularization to avoid overfitting.

CNN have 2 most important indicators:

- Accuracy: it's the fraction of predictions our model got right.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

- Loss: it's used to measure the inconsistency between predicted value (^y) and actual label (y). It is a non-negative value, where the robustness of model increases along with the decrease of the value of loss function. There are different formulas for the calculation of the loss.

TREE CNN

The modern world of digitized data produces new information every second, thus fueling the need for systems that can learn as new data arrives. Traditional deep neural networks are

static in that respect, and several new approaches to incremental learning are currently being explored.

Tree CNN is a methodology to create a CNN by means of a cascade of classificators. By using Tree CNN we can implement the so called “incremental learning”: after feeding the root node of the tree, the other branches are created on the base of the results by imposing some thresholds. In this way, the network can grow in depth or in width with the arrival of new data. Further details about our method will be later explained.

This idea was firstly developed by Deboleena Roy, from Purdue University, and her team [1]. In his work Tree-CNN is made of nodes connected as a tree. Each node represents a classifier and his outputs will be the input for the next node.

The Nodes could be divided into:

- Root Node: It is the first node of the tree that provides an initial classification of the input
- Branch Node: After the first classification, output are sent to next node that could be a branch node or a leaf node. The branch node has a parent and 1 or more children.
- Leaf Node: It is the final node of our classifier. It provides the final result of the predict process.

To avoid the “catastrophic forgetting”, the root Node is filled with new class images. Evaluating the output through comparison with pre choiced threshold, the network kept growing until is final, stable shape.

DETAILED DESCRIPTION OF USED METHODS

- **Step 0:**

Our team started by developing a Keras sequential net for the prediction of 5 classes.

Below is reported how the net is built, this particular structure has been created by looking to other coders in the literature, in the web and in Keras documentation. This represented a very first tentative to make things work. The ***Convolution2D*** layer represents the layer where convolution takes place, here the number of filters and their dimension is reported together to the activation function which is ‘relu’ in order to account for non-linearities in the net. The ‘input_shape’ is the img_shape to begin, then Sequential in Keras will think about the input shape in the next layers. The ***MaxPooling2D*** performs a max_pool with specified pool size. Usually Convolution starts with a number of filters of 32 to increase to 64 or 128. A ***Dropout*** layer is then performed in order to decrease the probability of overfitting. ***Flatten*** layer will flatten the input, so we’ll get a vector and then with ***Dense*** we are specifying that

we want fully connected layers. The last layer has to have the output dimensions equal to the number of classes we want to classify.

```
classifier = Sequential()
classifier.add(Convolution2D(32,4,4,input_shape=(224,224,3), activation ='relu'))
classifier.add(MaxPooling2D(pool_size=(3,3)))
classifier.add(Convolution2D(32,3,3, activation ='relu'))
classifier.add(MaxPooling2D(pool_size=(2,2)))
classifier.add(Convolution2D(64,2,2, activation ='relu'))
classifier.add(MaxPooling2D(pool_size=(2,2)))
classifier.add(Dropout(0.3))
classifier.add(Flatten())
classifier.add(Dense(output_dim=128, activation ='relu'))
classifier.add(Dense(output_dim=64, activation ='relu'))
classifier.add(Dense(output_dim= 32, activation ='relu'))
classifier.add(Dense(output_dim=5, activation ='sigmoid'))
```

We used Stochastic Gradient Descent as optimizer with a starting learning rate of 1e-4, a decay of the learning rate of 1e-6 after each update, a Nesterov momentum of 0.9 to accelerate the SGD in the relevant direction and dampens oscillations. We used 'sparse_categorical_crossentropy' as loss function for categorical data.

```
sgd = optimizers.SGD(lr=0.0001, decay=1e-6, momentum=0.9, nesterov=True)
classifier.compile(optimizer=sgd,
loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

- **Step 1: VGG16 net for the whole dataset**

After this very first tentative we've realized that Keras has many pre-built models and also many of them are pre-trained which means that the first weights aren't randomly initialized but downloaded from ***imagenet***. We tried the whole set of keras models and found out that ***vgg16*** is the model with best performances for our dataset.

```
base_model = keras.applications.vgg16.VGG16(include_top=False, weights='imagenet',
input_tensor=None, input_shape=(224, 224, 3))
```

From the base model we made some changes to the end of it by adding 4 layers that created our top model over the base network:

```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.2)(x)
x = Dense(self.n_classes, activation=activ_func)(x)
```

Also we added a list of callbacks which resulted very useful in training our nets.

earlyStopping *= keras.callbacks.EarlyStopping(monitor='val_acc', min_delta=0.001, patience=5, verbose=1, mode='auto')*

EarlyStopping is used to prevent overfitting and stop the training session when it's no more useful, allowing us to reduce gpu resources and saving time.

checkpointer *= keras.callbacks.ModelCheckpoint(monitor='val_acc', filepath='bestweightsmn.hdf5', verbose=1,save_best_only=True, save_weights_only=False, mode='auto', period=1)*

Checkpointer is used to save the state of the net every time an improvement is reached

reducelr *= keras.callbacks.ReduceLROnPlateau(monitor='val_acc', factor=0.1, patience=2, verbose=1, mode='auto', min_delta=0.001, cooldown=3, min_lr=0)*

ReduceLROnPlateau is used to reduce the LR when no further improvement is reached among epochs.

epoch_print_callback *= keras.callbacks.LambdaCallback(on_epoch_end = lambda epoch, logs: self.TG_bot.sendMessage(self.chat_id, "Epoch " + str(epoch) + "val_acc: " + str(logs['val_acc'])),on_train_end = lambda logs: self.TG_bot.sendMessage(self.chat_id, "Training is over!"))*

Moreover we created a ***personal callback*** which allowed us to get a notification on telegram everytime an epoch finished with the information of the validation accuracy and validation loss.

```
callbacks_list = [checkpointer, earlyStopping, reduceIr, epoch_print_callback]
```

Also we were able to get a notification when the training session were over with the final graphs of the accuracy and loss among the epochs for both training and test sets.

```
fig = plt.figure()
plt.plot(history.history['val_acc'])
plt.title(self.model_name+' accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
fig.savefig(self.model_name+'_acc.png')
plt.close()
f = open(self.model_name+'_acc.png',"rb")
self.TG_bot.sendPhoto(self.chat_id,f,caption="Accuracy of the model")
```

- **Step 2: Likelihood Matrix**

Once the net for the whole 5 classes was built we took the same model and made it work for only 2 classes (which are H, *Healthy* and AC, *AdenoCarcinoma*). Next we showed to the net a small percentage (10%) of pics of the other classes and we computed the likelihood matrix. This means we looked for each of the class the net had never seen what was the result that the net returned. Then we decided some thresholds based to the results and built new branches for the net. Each branch has been trained with the right dataset and then we computed the final accuracy of the obtained classifier. To compute the Likelihood matrix we were interested on the output of the net before the last layer.

```
outputs = K.function([model.layers[0].input],[model.layers[-2].output])
O.append(outputs([np.expand_dims(data.x_train[i], axis=0)])[0])
```

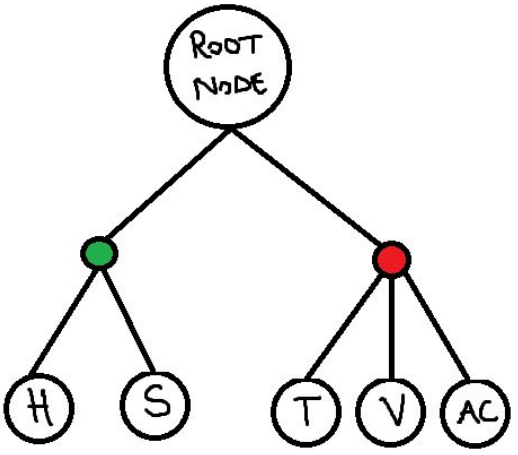
After having stored the outputs for 300 images (~10% of the training set) we computed the average on it getting the Likelihood matrix. Each row of it contains 2 values indicating the membership degree of that specific class for H and for AC

L.append(np.mean(O,0))

L_dict = {'AC':L[0],'H':L[1],'S':L[2],'T':L[3],'V':L[4]}

Key	Type	Size	Value
AC	float32	(2,)	[9.9923259e-01 7.6747703e-04]
H	float32	(2,)	[0.00149611 0.9985041]
S	float32	(2,)	[0.4472817 0.5527184]
T	float32	(2,)	[0.74986005 0.2501403]
V	float32	(2,)	[0.907776 0.09222431]

- **Step 3: Tree-CNN structure definition**

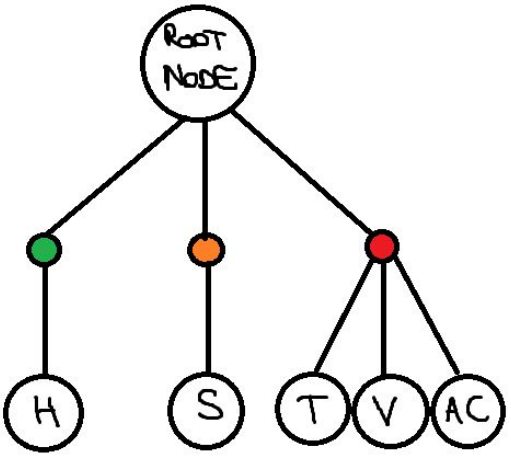


Based on the likelihood matrix we choice a ***threshold***, we had 2 main choices:

1) Threshold = 0.5 :
   - x > 0.5 → the net grows vertically and the element is grouped in the node with the highest likelihood. Later another classificator is needed to classify among the subclasses of the node

   PRO: the net grows vertically keeping a good shape

   CONS: It's very possible for an S element to finish in the wrong node and having a mistaken classification.



2) Another threshold such as 0.7 is added:
   - 0.5 < x < 0.7 → the net grows horizontally, the element isn't classified with enough accuracy in none of the child nodes, so another node is added in the upper level.
   - x > 0.7 → the net grows vertically.

PRO: It solves the misclassification of S elements.

CONS: Another horizontal node has been added, the net is not so far in shape from the direct “5 classes” classifier.

Choosing a threshold greater than 0.91 would have meant having directly the classifier for 5 classes.

- ## Step 4: Nets’ performances

After having analyzed pro and cons of the possibilities we tested the performances of both classifiers (using the script test.py) and as expected the method 2 gave us better results, above all for S elements. So we opted for method 2 with T1 = 0.5 and T2 = 0.7

In this way the TREE-CNN will be composed by an ACHS classifier and then if the element has a stronger membership for AC it’ll be redirected to the ACTV classifier which will perform a finer classification.

Then again using ‘test.py’ script we compared the chosen TREE-CNN with the 5-classes vgg16 classifier. We developed this script since it wasn’t possible to use the ‘evaluate’ method of keras for a cascade of classifiers such as our Tree-CNN. After many attempts the decision to perform data augmentation was taken. Data augmentation has been performed in order to get a new dataset where each class has 10000 images, the classical operations of data augmentation were taken.

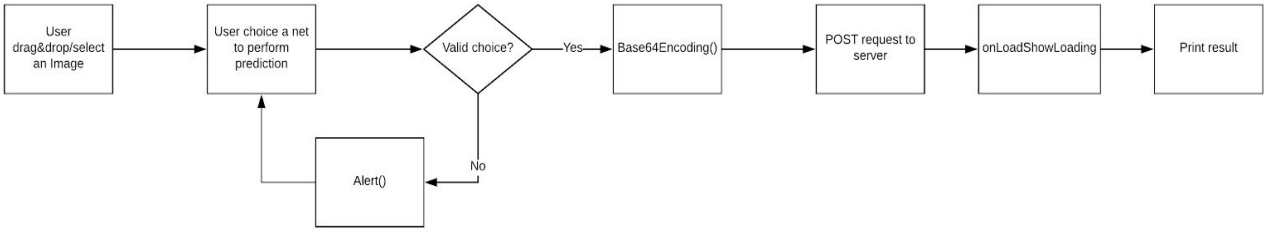
TOOLS

in order to provide a scalable service, we developed two tools in order to make the framework available also on devices with low hardware resources.

- **WEBSITE**

The website is composed by a WEBAPP working on a local server served by CherryPy Framework. The style of the site has been made using Bootstrap Templates and online templates example, acting on CSS code according to our needs. The contents are for the most part informative, but the core of the service is the technical part of the prediction. At the beginning we wanted to use the framework TensorFlowJS but because of some problems on the conversion of our starting Keras model into TF Model we opted for a different solution.

CLIENT SIDE



in the workflow above, we can see how the prediction page works communicating with server side. In the page the user has to select an image to be predicted. In our network we only work with images of dimension (224,224,3), so to obtain optimal performances, It'll better maintain the original image shape.

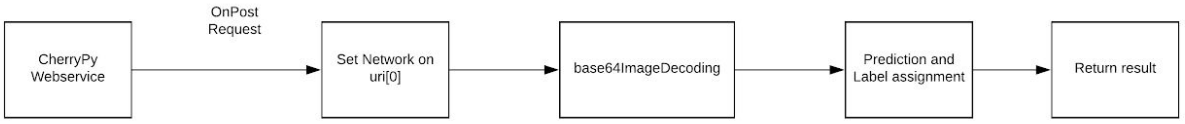
The selection of the network is **mandatory,** so to make prediction users have to select in the dropdown menu the right network.

The submit button performs some control on the environment:

1. The image has to be selected before pressing it
2. The field Network has to be selected

For the HTTP communication, the architecture uses a base64 mechanism of encoding, posting the payload produced using a POST request to the server client. After that the page rests in "listen mode", so It activates a loader icon in order to be more user friendly and when response of the request will be load, the content of the page will be changed showing the result.

SERVER SIDE



On the server side, we have a webservice using the python library CherryPy. The server works using parameters discrimination that on POST requests

1. Detects network to be used trough uri control. If the uri[1] will be made a prediction using VGG16 for 5 classes network, else the Tree CNN.

```
if uri[0]=="1":#give the type of net used
    typeofnet="VGG16 5 classes"
else:
    typeofnet="Tree CNN"
```

2. The payload containing the field "*typeofnet*" will be so passed to a custom made class called "*testing*" that performs prediction and gives label to the image.

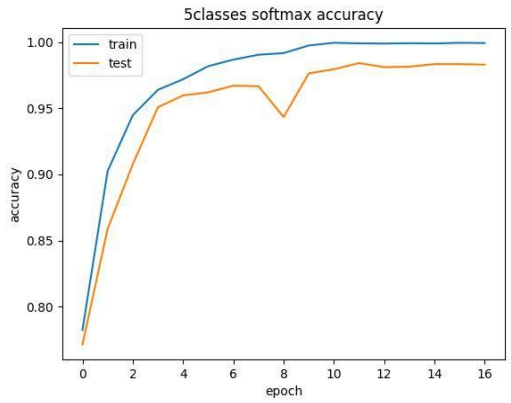
Before all of that we have the image encoding from base64 format to npy array.

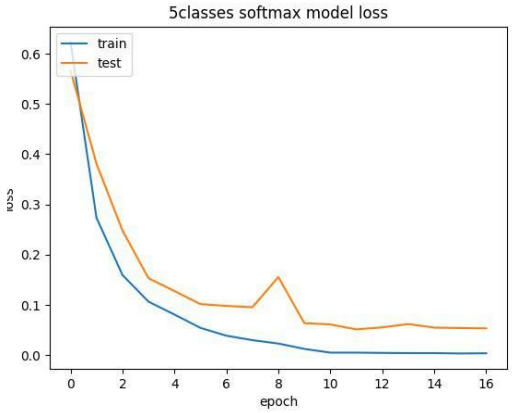
- **TELEGRAM BOT**

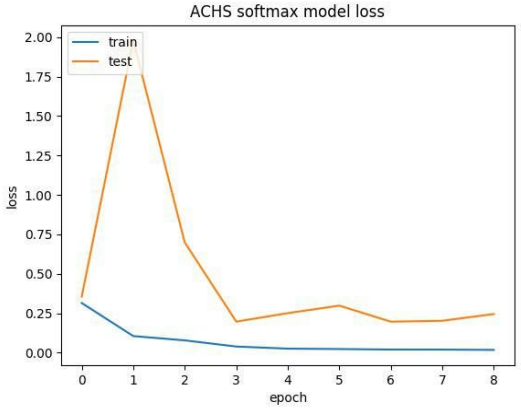
The bot is realized by using the telepot library. The bot by itself is intuitive and easy to use. It's thought to be used by doctors as immediate tool of screening. When you add the bot it'll ask you what kind of net you prefer to use to make the prediction, once selected it the bot will reply with a pop-up message saying that it's ready to take an image and make the prediction. So now all you need to do is to select the image and send it. Right now this is just a demonstrative tool, many things can be improved and the final result may be a direct link between the instrument which take the picture and the bot for an online diagnosis.
More over would be nice to improve the bot by adding the possibility of adding it to a group, thinking about it, in an hypothetical telegram group of doctors, a possible scenery would be: doc1 has to send an information to doc2 about patient X regarding the status of its carcinoma, doc1 could simply send the image by the bot with the prediction of the bot as caption of it.

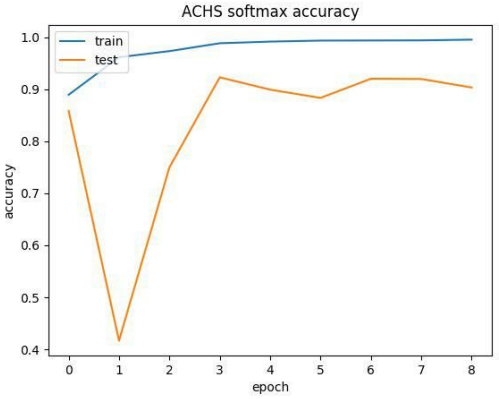
DETAILED DESCRIPTION OF THE RESULTS

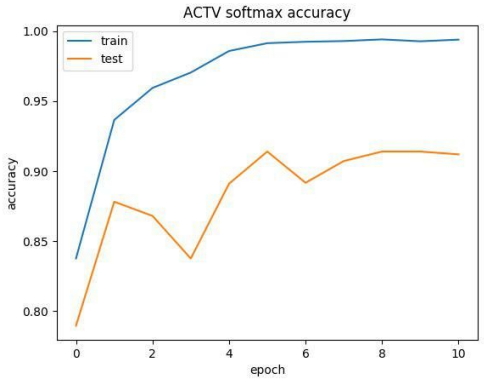
The graphs of the chosen nets are provided below:

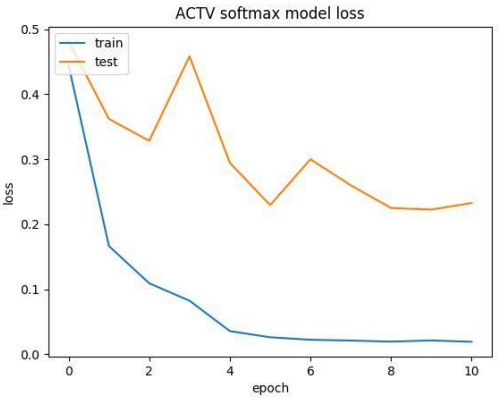












Discussion of the results for the main classifiers which are:

- 5 classes
- HAC (in this classifier has been computed the likelihood matrix)
- ACHS
- ACTV
- TREE-CNN

For the evaluation of our model, we used the default test set.

For the ACH dataset

x_test	(2000, 224, 224, 3)	RGB images
y_test	(2000)	labels (0-AD/AC, 1-H)

and for the STV dataset.

x_test	(1619, 224, 224, 3)	RGB images
y_test	(1619)	labels (0-S, 1-T, 2-V)

In order to recreate our right image pattern to perform evaluation, we merged all together the test sets for the 5 classes evaluation and we created ACHS and ACTV for the Tree CNN Classification.

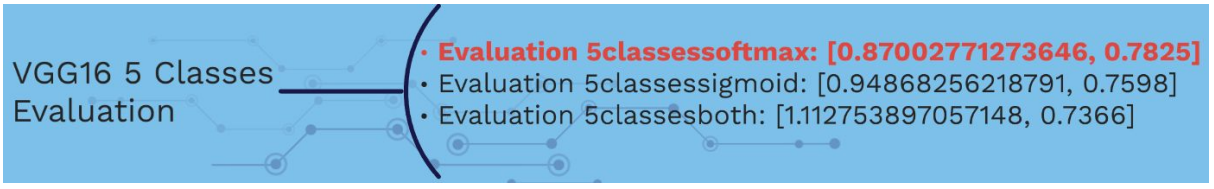
For the performances comparison we used different output layers:

1. Sigmoid
2. Softmax
3. Sigmoid and Softmax

This choice was led by the fact that to perform network feeding (See Method chapter), it's more solid to use a sigmoid because it has not value normalization like softmax. Having to make a mean between the predicted values of an image batch, we did not consider initially the output of the last layer (softmax) but the one referred to sigmoid that is the layer before the last one.

For the construction of the final prototype, all alternatives have been considered in order to perform the best choice possible.

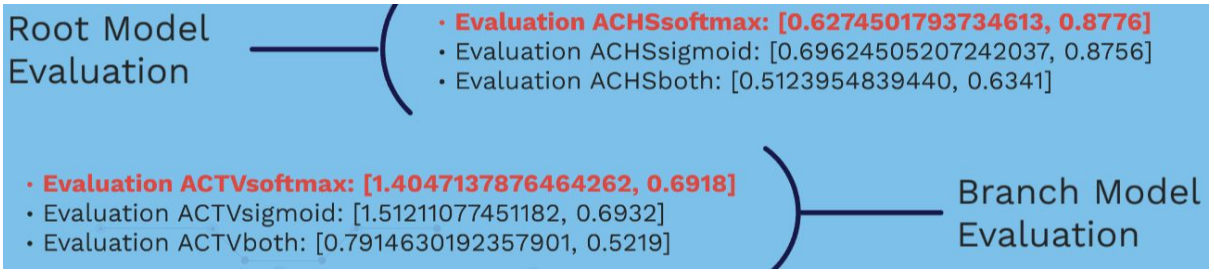
Working on *5 Classes Network* we have used the function model.evaluate() after having compiled the model itself.



In the table above, there is the evaluation for the models proposed. the first value is the loss, while the second one represents the accuracy.

As we can see the highest performance values are in the network using just softmax layer as output.

After that, we split our Tree CNN, evaluating root model and branch model separately.



We remind that the root Model is the 3 classes VGG16 based on AC-H-S, while the branch model is based on AC-T-V.

In both cases the best choice remains the usage of softmax as only output layer.

The accuracy is very high on the root model while it is lower on the branch model evaluation. This could be referred to the fact that AC-T-V classes are very similar among them, representing borderline situation, so for this reason classifier cannot distinguish them suitably.

After this first phase, allowing team to choose the appropriate architecture, we set up an evaluation script that has been discussed on the method chapter allowing the evaluation of Tree CNN in terms of accuracy.

We obtained the following results:

```
Prestazioni TREE-CNN:
AC: 397/1000
H:  928/1000
S:  451/500
T:  291/561
V:  482/558
Total Accuracy:  0.7043382149765128

Prestazioni VGG16:
AC: 498/1000
H:  953/1000
S:  412/500
T:  438/561
V:  531/558
Total Accuracy:  0.7825366123238464
```

As we can see, the AC class is bad classified by both models.

	ACCURACY- TREE CNN	**ACCURACY- 5 CLASSES**
AC	0.4	0.5
H	0.92	0.95
S	0.90	0.82
T	0.51	0.78
V	0.86	0.95

Tree CNN shows generally lower accuracy values in all classes classification. Exception for S that has a greater accuracy in tree.
According to this analysis and by looking at the total accuracy, the best net is the 5 classes CNN and the development of the algorithm in our case, using our own-development method did not give enchantment respect to the classic approach.

FUTURE OUTLOOKS

Many things can obviously be improved:
1) The net as built is only capable to classify images with shape (224,224,3) which obviously is not always the case of every medical image.
2) The net itself can't actually grow automatically, if new classes want to be added some code should change, we can say it's a semiautomatic grow since there's need of an operator to compute the new likelihood and to choose the new thresholds, after that the tree can grow. Obviously with time, experience and why not money, it's possible to arrange the code in a total automatic way. This is only a starter point.
3) The performances of the TREE-CNN aren't better of the net for the direct classification of each class. This may be because of the restricted number of classes which make the direct classifier a strong competitor and also because of the strong similarity of images among some classes.
4) Implementation of IoT procedures to link medical instrumentation directly to the server where the net is stored.

In a not too far future could be possible for doctors to have a difficult diagnosis as simple as taking a picture.

Sincerely, the Team.
Francesco Alotto, Riccardo Di Dio, Ylenia Placanica

BIBLIOGRAPHY

[1]: *Deboleena Roy, Priyadarshini Panda, Kaushik Roy*, Tree-CNN: A Hierarchical Deep Convolutional Neural Network for Incremental Learning