

OR Project

Abanoub Ashraf Ezzat Zaki ID: 1
Eman Rafik ID: 13
Bassam Rageh Ibrahim Ali ID: 15
Toka Alaa Ahmed ID:16
Reham Mohamed Naguib ID: 19

Overview

Magnetic resonance imaging (MRI) has been one of the most effective ways of detecting knee injuries. In synchronous with the improvements achieved in deep learning and image classification, researchers have gone towards computer-aided solutions for interpreting MRI images. These solutions have highly aided radiologists in diagnosis and somehow fasten the process of interpretation which was time-intensive.

This study is mainly around the MRNet model proposed in the research article 'Deep-learning-assisted diagnosis for knee magnetic resonance imaging: Development and retrospective validation of MRNet'. Firstly, we started by re-building the proposed model. Secondly, we built three well-known neural networks: Visual Geometry Group (VGG), Residual Neural Network (ResNet), and Inception V3. We experimented replacing the feature extractor part of the proposed model (based on AlexNet) by these three networks alternatively.


A transfer learning approach was then used, where the weights of inception V3 network were initialized by the weights of ImageNet using Keras built-in model

MRNET Paper

Summary

In the MRNet paper, they proposed an automated system to detect anterior cruciate ligament(ACL) tears, meniscal tears and abnormalities in magnetic resonance imaging (MRI) exams of the knee. The proposed system is a deep learning model trained on the MRNet dataset. The data set is Digital Imaging and Communication in Medicine (DICOM) and clinical reports for knee MRI exams performed at Stanford Medical Center. Three types of series for each exam were used in the proposed experiment: sagittal plane T2-weighted series, coronal plane T1-weighted series, and axial plane PD-weighted series. Each series has slices ranging between 17 and 61. The dataset was split into training, tuning and validation sets, with each set containing at least 50 exams having a true label for each of: abnormality, ACL tears and meniscal tears.

The system has three models one for predicting each of ACL tears, meniscal tears and abnormalities. Each individual model is composed of three blocks of MRNet, which is a convolutional neural network that starts with a feature extractor based on AlexNet followed by an average pooling layer then a maximum pooling layer. The final layer is a dense layer with a sigmoid activation function and one output perceptron to give the output probability. Each of the three MRNet blocks predicts a probability given an input



series of one type (sagittal, coronal or axial), then the three probabilities are passed through a logistic regression model to give one output probability for each examination. In the proposed paper, they applied a transfer learning approach due to the limitations in the training data size, where the weights of AlexNet are initially set to the optimal weights fitting the ImageNet database. Weights were then fine-tuned to fit the MRNet dataset, where they trained each model using binary cross-entropy loss and backpropagation algorithm.

The described experiment starts by extracting images from DICOM files then performing image pre-processing. Images were scaled to 256 x 256 pixels, converted to Portable Network Graphics (PNG) format and subjected to intensity standardization. Finally, image pixels were adjusted to the standard pixels range of PNG images. For the training set, they applied data augmentation where each example was randomly rotated, shifted and flipped horizontally. Also, the actual output of training examples was extracted from the clinical reports to be fed to the model. The input of each MRNet block has dimension $s \times 3 \times 256 \times 256$, where s is a variable parameter representing the number of images in the input series and 3 is for color channels. Since the input image has no color channels, each image is replicated three times to fit the input dimension. The feature extractor is fed with the input to give a tensor of dimension $s \times 256 \times 7 \times 7$ having the feature map of each image in the series, which is then passed through the average pooling layer to reduce the feature maps' dimension to $s \times 256$. Finally, a global average is applied to all series images to give a 256-dimension vector that is passed to the final dense layer and predicts the output probability for the specified type.

The model was evaluated on an internal validation set (test set) labeled by MSK radiologists and it achieved accuracies of 0.85, 0.867 and 0.725 for general abnormalities, ACL tears and meniscus tears, respectively. The area under the receiver operating characteristic curve (AUC) for the three injuries, respectively, were 0.937, 0.965 and 0.847. Furthermore, the trained model was evaluated on an external dataset with sagittal exams of ACL injuries and it achieved 0.824 AUC.

The future work is to improve the model performance and generalize deep learning models for MRI.

Results

In this study, we re-built the MRNet model using TensorFlow, Keras. We achieved accuracies of 0.792, 0.6917 and 0.617 for abnormalities, ACL tears and meniscus tears, respectively.

Loading Data

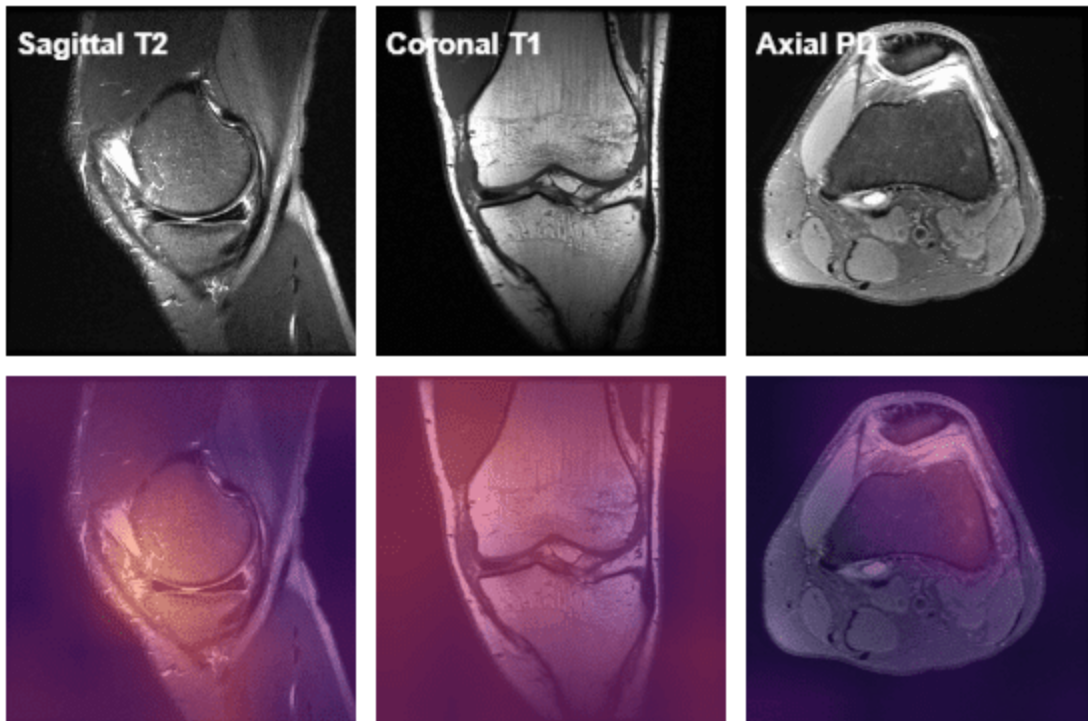
The input to MRNet has dimensions $s \times 3 \times 256 \times 256$, where s is the number of images in the MRI series and 3 is the number of color channels.

Since data is black and white, each image in the MRI series is repeated 3 times to obtain the extra dimension of the color channels.

Data is divided into 2 main parts training which are 1130 cases and validation which are 120 cases.

So we divided the training part 90% to be training data of 1017 cases and the rest 10% to be the validation data of 113 cases.

So the main validation cases which are 120 cases will be the testing data.



Implementation of keras data generator:

```
class data_generator(keras.utils.Sequence):
    def __init__(self, injury, plane, data_type):
        self.injury = injury
        self.plane = plane
        self.data_type = data_type
        if(data_type == 'train'):
            self.labels = pd.read_csv('train-{}.csv'.format(self.injury), names=['case', 'label'], header=None,
                                     dtype={'case': str, 'label': np.int64})['label'].tolist()
            self.case_list = pd.read_csv('train-{}.csv'.format(self.injury), names=['case', 'label'], header=None,
                                     dtype={'case': str, 'label': np.int64})['case'].tolist()
            self.labels = self.labels[0:1017]
            self.case_list = self.case_list[0:1017]
            self.data_path = 'train'
        elif(data_type == 'valid'):
            self.labels = pd.read_csv('valid-{}.csv'.format(self.injury), names=['case', 'label'], header=None,
                                     dtype={'case': str, 'label': np.int64})['label'].tolist()
            self.case_list = pd.read_csv('valid-{}.csv'.format(self.injury), names=['case', 'label'], header=None,
                                     dtype={'case': str, 'label': np.int64})['case'].tolist()
            self.labels = self.labels[1017:1130]
            self.case_list = self.case_list[1017:1130]
            self.data_path = 'train'
        else:
            self.labels = pd.read_csv('valid-{}.csv'.format(self.injury), names=['case', 'label'], header=None,
                                     dtype={'case': str, 'label': np.int64})['label'].tolist()
            self.case_list = pd.read_csv('valid-{}.csv'.format(self.injury), names=['case', 'label'], header=None,
                                     dtype={'case': str, 'label': np.int64})['case'].tolist()
            self.data_path = 'valid'

    def __len__(self):
        return len(self.case_list)

    def __getitem__(self, index):
        fpath = '{}/{}/{}/.npy'.format(self.data_path, self.plane, self.case_list[index])
        stack = np.load(fpath).astype(float)
        for i in range(stack.shape[0]):
            stack[i] = stack[i]/255.
        stack = np.repeat(stack[:, :, :, np.newaxis], 3, axis=3)
        stack = np.expand_dims(stack, axis=0)

        y = self.labels[index]
        y = np.array(y).reshape(1,1)
        return stack, y
```

It takes data type ('train', 'valid', 'test'), injury type ('ACL', 'abnormal', 'meniscus') and the plane required ('axial', 'coronal', 'sagittal').

In the initialization function, it prepares the cases of the required data and its outputs.

In the get item function, it takes the index of the required batch and returns stack and y.

Where stack is the input which is a batch of size one each time called, whose dimensions is $(1 \times s \times 256 \times 256 \times 3)$ and y is the output of this case with the specified injury.

Feature Extraction Networks

Visual Geometry Group (VGG)

The VGG model uses 5 stacks of convolutional layers called VGG blocks. The first 2 layers have 2 convolutional layers and the next 3 have 3 convolutional layers. Each block ends with a max-pooling layer with a 2×2 window and 2 strides. The convolutional layers use the ReLU activation function.

The convolutional layers use 3×3 filters with 1×1 stride. The blocks have 64, 128, 256, 512, 512 filters in each layer respectively.

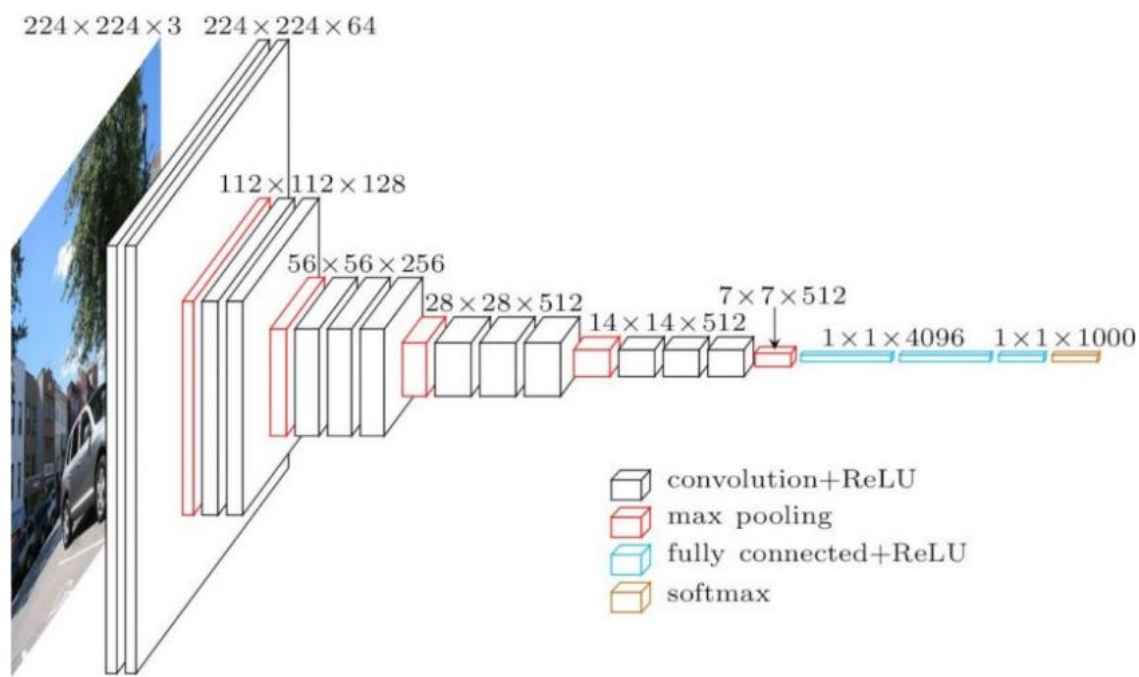
The output of the VGG model is passed to a global average pooling then a global max pooling before entering a fully connected layer with sigmoid activation to output the final probability.

Results

Test accuracy for abnormal: 0.7916

Test accuracy for ACL: 0.55

Test accuracy for meniscus: 0.5666

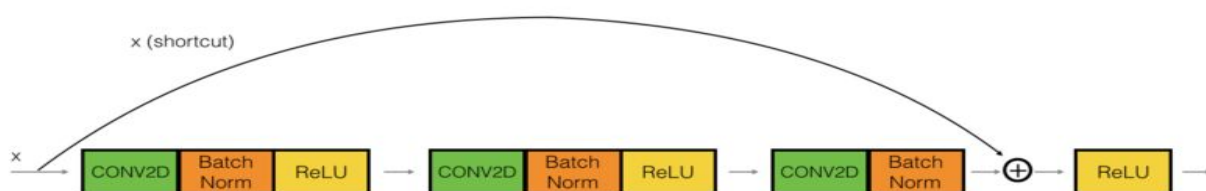


VGG16 Architecture

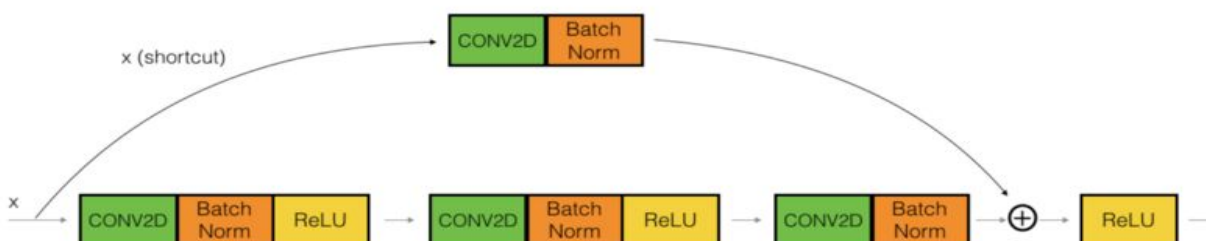
Residual Neural Network (ResNet)

ResNet-50 is a deep convolutional residual neural network. This type of network can benefit from increasing the depth of the network and enjoy the accuracy gain, while on the other hand, the "plain" network performance decreases when we increase its depth. It consists of two basic blocks depending on the input layer dimensions, the identity block is used when the dimensions of the input match the dimensions of the output, it consists of 3 components each consisting of Conv2D layers, batch normalization and ReLU activation layer except for the third component. A short cut is added between the input and the output. The convolutional block consists of the same three components except that the shortcut path also contains a Conv2D and a batch normalization layer to resize the input dimensions to match the output dimensions. A ReLU activation is then applied on the main path. ResNet-50 consists of 50 layers. The architecture below shows these layers using the conv block and the identity block.

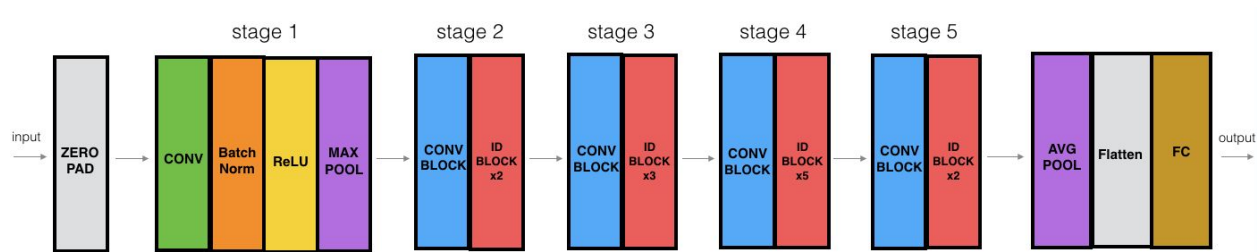
Identity block:



Convolutional block:



Architecture:



Results

Test accuracy for abnormal: 0.7917

Test accuracy for ACL: 0.5500

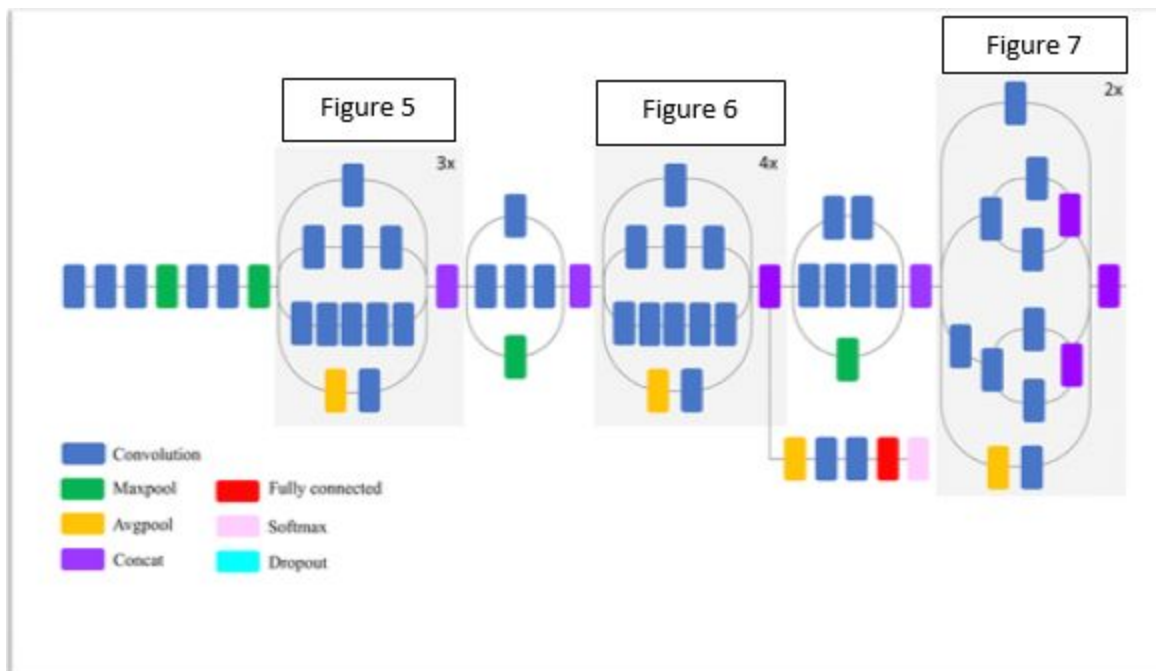
Test accuracy for meniscus: 0.5667

Inception V3

We found many implementations with slight variations such as the number of filters, the number of blocks, the kernel size, and batch normalization and activation layers' order. We used the model implemented in Keras version 2.3.1 as a reference.

A paper called “Rethinking the Inception Architecture for Computer Vision” was used as a reference when building the inception blocks. Each block is created by a function that has the same name as the figure used to implement it.

Model structure:



Note in implementation

Blue color represents: Convolution layer + Batch Normalization layer + “ReLU” Activation layer

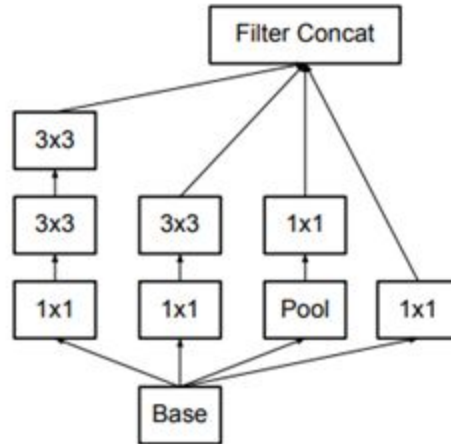


Figure 5. Inception modules where each 5×5 convolution is replaced by two 3×3 convolution, as suggested by principle 3 of Section 2.

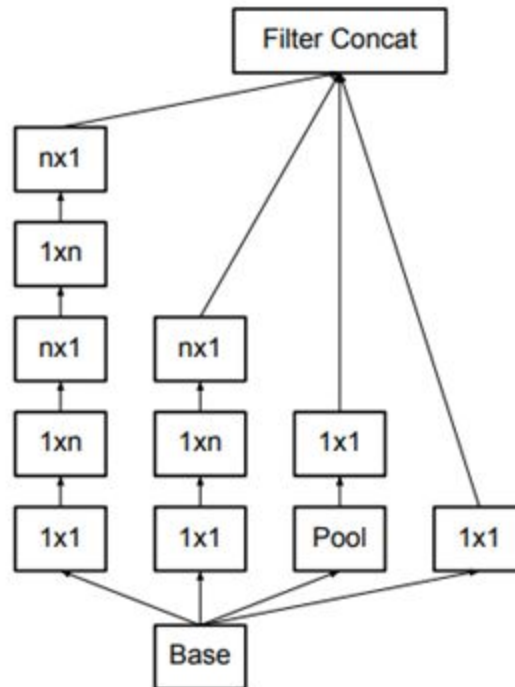


Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ for the 17×17 grid. (The filter sizes are picked using principle 3)

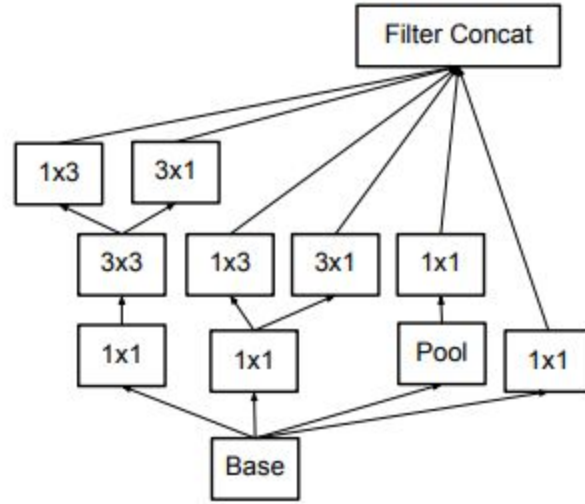


Figure 7. Inception modules with expanded the filter bank outputs. This architecture is used on the coarsest (8×8) grids to promote high dimensional representations, as suggested by principle 2 of Section 2. We are using this solution only on the coarsest grid, since that is the place where producing high dimensional sparse representation is the most critical as the ratio of local processing (by 1×1 convolutions) is increased compared to the spatial aggregation.

The input to inception model has original dimensions $s \times 299 \times 299 \times 3$ so we had to adapt input dimensions of the model to match with the dataset dimensions $s \times 256 \times 256 \times 3$. First, each 2-dimensional MRI image slice is passed through a feature extractor to obtain an $s \times 6 \times 6 \times 2048$ tensor containing features for each slice. A global average pooling layer is then applied to reduce these features to $s \times 2048$. We then applied max-pooling across slices to obtain a 2048-dimensional vector, which is passed to a fully connected layer to obtain a prediction probability.

Results

Test accuracy for abnormal: 0.95

Test accuracy for ACL: 0.55

Test accuracy for meniscus: 0.679

Transfer Learning

Transfer learning generally refers to a process where a model trained on one problem is used in some way on a second related problem.

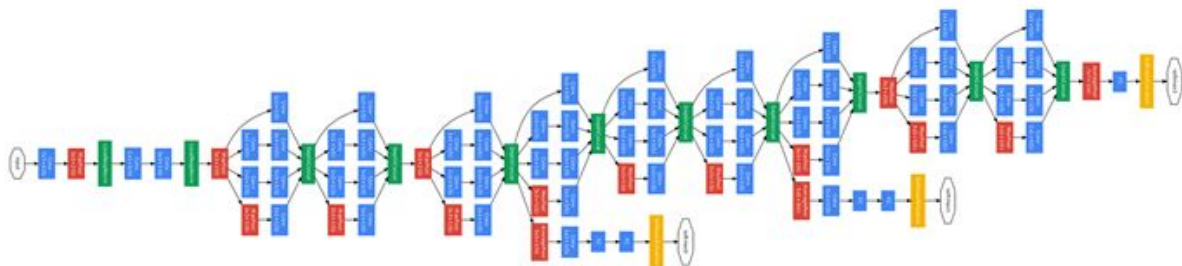
In deep learning, transfer learning is a technique whereby a neural network model is first trained on a problem similar to the problem that is being solved. One or more layers from the trained model are then used in a new model trained on the problem of interest.

Instead of training the models starting from a random state, we can use transfer learning to start with a better set of networks' weights trained on famous datasets.

Here, we will use ImageNet weights to enhance the training of the Inception network.

We have chosen the best-performed network from the previous part and applied transfer learning with ImageNet weights.

Since inception_network has had the best results in testing, so we will use the **InceptionV3** network model built-in Keras.



Results

Test accuracy for abnormal: 0.7917

Test accuracy for ACL: 0.55

Test accuracy for meniscus: 0.5583

Contribution

We proposed building a model that takes the predictions from VGG, Resnet, and Inception models and passes them to logistic regression to generate a single output for each view. The produced three views are then passed to another logistic regression to produce single output for each exam.

Conclusion

The dataset is very small which leads to overfitting. This can be solved by gathering more samples or using data augmentation.

The accuracy stops increasing after 3 - 5 epochs. It could be a result of a large learning rate. So changing the learning rate may solve the problem.

Google Colab Link

https://colab.research.google.com/drive/19FS3siYAP_03xP83VUo8HI6C81iEH7ct?usp=sharing

References

General

- <https://journals.plos.org/plosmedicine/article?id=10.1371/journal.pmed.1002699>
- https://keras.io/guides/functional_api/
- https://www.ahmedbesbes.com/blog/acl-tear-detection-part-1?fbclid=IwAR1Oa6uyz5VaFUt-HvxaslaEGxNlaudzaHPEcz7bp_2ZQt_uEhJE6veQn6s

Loading data

- <https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly#previous-situation>

Alexnet

- <https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
- <https://engmrk.com/alexnet-implementation-using-keras/>

- <https://www.learnopencv.com/understanding-alexnet/>

VGG

- <https://neurohive.io/en/popular-networks/vgg16/>
- <https://machinelearningmastery.com/how-to-implement-major-architecture-innovations-for-convolutional-neural-networks/?fbclid=IwAR0u1h9UhZXCHTniCHJiwdV8T4T9eMdKNGNRklSyBgV93RgnbSdqfUI2Dag>

Inception

- https://arxiv.org/pdf/1512.00567v3.pdf?fbclid=IwAR0UOHUWUJLPUKLeABK0jTJVyiwnAHH6iiufMhN1E_woG6mb9-IQDIEPNko
- <https://josephpcohen.com/w/wp-content/uploads/inception-v3.pdf>
- https://gist.github.com/neggert/f8b86d001a367aa7dde1ab6b587246b5?fbclid=IwAR0tA-Gzase2K_S09PzZvLjjrnXfAimlNS1wbqIYEIP9F2Cj2dep64Z_9R0
- https://github.com/fchollet/deep-learning-models/blob/master/inception_v3.py

ResNet

- <https://arxiv.org/abs/1512.03385>
- <https://pylessons.com/Keras-ResNet-tutorial/>
- [http://ethen8181.github.io/machine-learning/keras/resnet_cam/resnet_cam.html#ResNet-\(Residual-Network\)](http://ethen8181.github.io/machine-learning/keras/resnet_cam/resnet_cam.html#ResNet-(Residual-Network))
- <https://machinelearningmastery.com/how-to-implement-major-architecture-innovations-for-convolutional-neural-networks/>

Transfer learning

- https://www.tensorflow.org/api_docs/python/tf/keras/applications/InceptionV3
- <https://alexisbcook.github.io/2017/using-transfer-learning-to-classify-images-with-keras/>