



Operation Research Final Project



**ALEXANDRIA UNIVERSITY
FACULTY OF ENGINEERING**

**COMPUTER AND SYSTEMS ENGINEERING
DEPARTMENT**

Team Members:

Mohamed Said Ibrahim Mohamed Bdr id: 57

Mohamed AboBakr Al-Seddik Hassan id: 51

Part 1: MRNET Paper Summary:

Background

Magnetic resonance imaging (MRI) of the knee is the preferred method for diagnosing knee injuries. However, interpretation of knee MRI is time-intensive and subject to diagnostic error and variability. In this study we developed a deep learning model for detecting general abnormalities and specific diagnoses.

What we want to achieve from this study?

1. We wanted to determine whether a deep learning model could improve the diagnostic accuracy, specificity, or sensitivity of clinical experts, including general radiologists and orthopedic surgeons.
2. We wanted to see if a deep learning model could succeed in the clinically important task of detecting disorders in knee magnetic resonance imaging (MRI) scans.

What did the researchers do and find and what do these findings mean?

1. We experimented with providing model outputs to general radiologists and orthopedic surgeons during interpretation and observed statistically significant improvement in diagnosis of ACL tears with model assistance.
2. Our deep learning model predicted 3 outcomes for knee MRI exams (anterior cruciate ligament [ACL] tears, meniscal tears, and general abnormalities) in a matter of seconds and with similar performance to that of general radiologists.
3. When externally validated on a dataset from a different institution, the model picked up ACL tears with high discriminative ability.
4. Deep learning has the potential to provide rapid preliminary results following MRI exams and improve access to quality MRI diagnoses in the absence of specialist radiologists.
5. Providing clinical experts with predictions from a deep learning model could improve the quality and consistency of MRI interpretation



Introduction

Magnetic resonance imaging (MRI) of the knee is the standard-of-care imaging modality to evaluate knee disorders.

Due to the quantity and detail of images in each knee MRI exam, accurate interpretation of knee MRI is time-intensive and prone to inter and intra-reviewer variability.

An automated system for interpreting knee MRI images has a number of potential applications, such as quickly prioritizing high-risk patients in the radiologist workflow and assisting radiologists in making diagnoses.

Deep learning approaches, in being able to automatically learn layers of features, are well suited for modeling the complex relationships between medical images and their interpretations.

In this study, we present MRNet, a fully automated deep learning model for interpreting knee MRI, and compare the model's performance to that of general radiologists.

Dataset

A dataset of 1,370 knee MRI examinations. The dataset contained 1,104 (80.6%) **Abnormal**

exams, with 319 (23.3%) **ACL** tears and 508 (37.1%) **Meniscal** tears.

ACL tears and **Meniscal** tears occurred concurrently in 194 (38.2%) exams.

The exams were split into a Training set (1,130 exams, 1,088 patients), a Tuning set (120 exams, 111 patients), and a Validation set (120 exams, 113 patients).

To form the Validation and Tuning sets, stratified random sampling was used to ensure that

at least 50 positive examples of each label (**Abnormal**, **ACL** tear, and **Meniscal** tear) were present in each set.

All exams from each patient were put in the same split.

Discussion

The purpose of this study was to design and evaluate a deep learning model for classifying pathologies on knee MRI and to compare performance to human clinical experts both with and without model assistance during interpretation in a crossover design. Our results demonstrate that a deep learning approach can achieve high performance in clinical classification tasks on knee MR, with AUCs for abnormality detection, ACL tear detection, and meniscus tear detection of 0.937 (95% CI 0.895, 0.937), 0.965 (95% CI 0.938, 0.965), and 0.847 (95% CI 0.780, 0.847), respectively. Notably, the model achieved high specificity in detecting ACL tears on the internal validation set, which suggests that such a model, if used in the clinical workflow, may have the potential to effectively rule out ACL tears.

Table 3. Comparison of unassisted and model-assisted performance metrics of clinical experts on the validation set.

Metric	Abnormality		ACL tear		Meniscal tear	
	Mean difference (95% CI)	<i>p</i> -Value <i>q</i> -value	Mean difference (95% CI)	<i>p</i> -Value <i>q</i> -value	Mean difference (95% CI)	<i>p</i> -Value <i>q</i> -value
Specificity	0.026 (−0.026, 0.079)	0.138 0.248	0.048 (0.029, 0.068)	<0.001 0.006	−0.006 (−0.035, 0.024)	0.667 0.692
Sensitivity	0.020 (−0.022, 0.062)	0.150 0.253	−0.004 (−0.041, 0.033)	0.592 0.639	0.052 (−0.002, 0.105)	0.028 0.110
Accuracy	0.021 (−0.008, 0.051)	0.069 0.173	0.023 (0.001, 0.045)	0.020 0.092	0.023 (−0.011, 0.057)	0.077 0.173

Mean differences (95% CIs) in clinical experts' performance metrics (model-assisted minus unassisted) for abnormality, anterior cruciate ligament (ACL) tear, and meniscal tear detection. Increases in performance when provided model assistance were assessed with a 1-tailed *t* test on the individual differences; both unadjusted *p*-values and adjusted *q*-values are reported. A *q*-value < 0.05 indicates statistical significance. Individual differences in performance metrics provided in [S2 Table](#).

<https://doi.org/10.1371/journal.pmed.1002699.t003>

	TP	FP	TN	FN	TPR	FPR	PPV	ACC
CLR	12	5	7	12	0.500	0.417	0.706	0.528
ARACNE	7	3	9	17	0.292	0.250	0.700	0.444
MRNET	17	6	6	7	0.708	0.500	0.739	0.639
MI3	9	5	7	15	0.375	0.417	0.643	0.444
MIDER	—	—	—	—	—	—	—	—
PCA-PMI	19	3	9	5	0.792	0.250	0.864	0.778
RRMRNET	10	2	10	14	0.417	0.167	0.833	0.556

Model

Preprocessing. Images were extracted from Digital Imaging and Communications in Medicine (DICOM) files, scaled to 256×256 pixels, and converted to Portable Network Graphics (PNG) format using Python.

MRNet. The primary building block of our prediction system is MRNet, a convolutional neural network (CNN) mapping a 3-dimensional MRI series to a probability.

The input to MRNet has dimensions $s \times 3 \times 256 \times 256$, where s is the number of images in the MRI series (3 is the number of color channels). First, each 2-dimensional MRI image slice was passed through a feature extractor based on AlexNet to obtain a $s \times 256 \times 7 \times 7$ tensor containing features for each slice.

A global average pooling layer was then applied to reduce these features to $s \times 256$. We then applied max pooling across slices to obtain a 256-dimensional vector, which was passed to a fully connected layer and sigmoid activation function to obtain a prediction in the 0 to 1 range. We optimized the model using binary cross-entropy loss.

Training a CNN for image classification from scratch typically requires a dataset larger than 1,130 examples. For this reason, we initialized the weights of the AlexNet portion of the MRNet to values optimized on the ImageNet database.

Combining MRNet predictions. Given predictions from the sagittal T2, coronal T1, and axial PD MRNets on the training set, along with their corresponding original labels, we trained a logistic regression to weight the predictions from the 3 series and generate a single output for each exam

Definitions for labels were as follows:

Abnormality: normal (all images reviewed are free of abnormalities) or abnormal (the abnormal findings in the internal validation set that were not ACL tear or meniscal tear)

ACL: intact (normal, mucoid degeneration, ganglion cyst, sprain) or tear (low-grade partial tear with $<50\%$ of fibers torn, high-grade partial tear with $>50\%$ of fibers torn, complete tear).

Meniscus: intact (normal, degenerative changes without tear, postsurgical changes without tear) or tear (increased signal reaching the articular surface on at least 2 slices or morphologic deformity).

Model performance

For abnormality detection, ACL tear detection, and meniscal tear detection, the model achieved AUCs of 0.937 (95% CI 0.895, 0.980), 0.965 (95% CI 0.938, 0.993), and 0.847 (95% CI 0.780, 0.914), respectively.

The model achieved a sensitivity of 0.879 (95% CI 0.800, 0.929) and accuracy of 0.850 (95% CI 0.775, 0.903), while the general radiologists achieved a sensitivity of 0.905 (95% CI 0.881, 0.924) and accuracy of 0.894 (95% CI 0.871, 0.913).

The model was highly specific for ACL tear detection, achieving a specificity of 0.968.

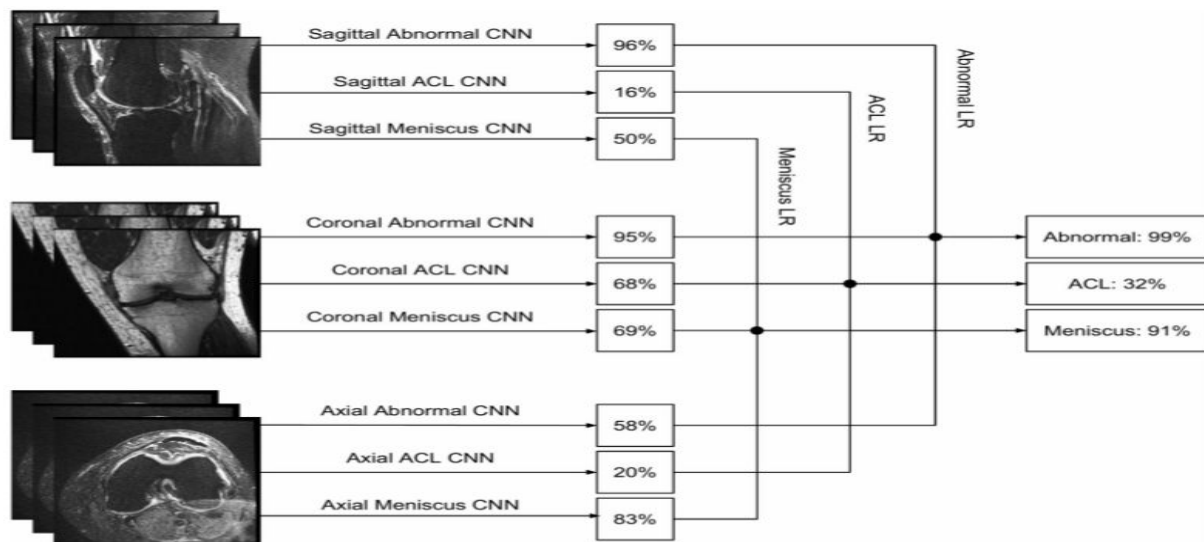


Table 2. Comparison of model and general radiologists on the validation set.

Prediction	Specificity (95% CI)	p-Value q-value	Sensitivity (95% CI)	p-Value q-value	Accuracy (95% CI)	p-Value q-value
Abnormality						
Model, threshold = 0.5	0.714 (0.500, 0.862)	—	0.879 (0.800, 0.929)	—	0.850 (0.775, 0.903)	—
Unassisted general radiologist micro-average	0.844 (0.776, 0.893)	0.247 0.344	0.905 (0.881, 0.924)	0.528 0.620	0.894 (0.871, 0.913)	0.201 0.301
ACL tear						
Model, threshold = 0.5	0.968 (0.890, 0.991)	—	0.759 (0.635, 0.850)	—	0.867 (0.794, 0.916)	—
Unassisted general radiologist micro-average	0.933 (0.906, 0.953)	0.441 0.566	0.906 (0.874, 0.931)	0.002 0.019	0.920 (0.900, 0.937)	0.075 0.173
Meniscal tear						
Model, threshold = 0.5	0.741 (0.616, 0.837)	—	0.710 (0.587, 0.808)	—	0.725 (0.639, 0.797)	—
Unassisted general radiologist micro-average	0.882 (0.847, 0.910)	0.003 0.019	0.820 (0.781, 0.853)	0.504 0.619	0.849 (0.823, 0.871)	0.015 0.082

The model was compared to unassisted general radiologists in detection of abnormality, anterior cruciate ligament (ACL) tear, and meniscal tear on a validation set of 120 knee MRI exams on which the majority vote of 3 musculoskeletal radiologists serves as the reference standard. A threshold of 0.5 was used to convert model probabilities to binary predictions before computing specificity, sensitivity, and accuracy. We use 95% Wilson score confidence intervals to estimate the variability in specificity, sensitivity, and accuracy estimates. We conducted a 2-sided Pearson's chi-squared test to evaluate whether there was a difference between the model and the micro-average of unassisted general radiologists. For each task and metric, we report both unadjusted *p*-values and adjusted *q*-values from this test. A *q*-value < 0.05 indicates statistical significance.

Questions & Answers And Main Definitions:

What is Keras and why to use it?

Keras is a high level API [open-source neural-network library written in Python], it can run on top of tensorflow and it is fast and user friendly and **Why** so you can have fast experimentation with deep neural networks without knowing everything about tensorflow.

What is neural network and why to use it?

The basic idea behind a neural network is to simulate (copy in a simplified but reasonably faithful way) lots of densely interconnected brain cells inside a computer so you can get it to learn things, recognize patterns, and make decisions in a humanlike way and **why** to solve complex problems people face in real-world situations. These networks can learn and model the relationships between inputs and outputs that are complex and nonlinear. We use a neural network to improve decision processes in areas such as: Medicare fraud detection and credit card.

What is meant by a dense layer?

Dense layer is a **fully connected layer**, meaning all the neurons in a **layer** are **connected** to those in the next **layer**.

What is the Sequential Model?

A Sequential model is appropriate for a **plain stack of layers** where each layer has **exactly one input tensor and one output tensor**, it shows the calculation order of the layers.

What is the Loss/Optimizer used for?

They are mainly used to find the best model.

Optimizers update the weight parameters to minimize the loss function. Loss function acts as guides to tell the **optimizer** if it is moving in the right direction to reach the required performance.

Loss is the difference between calculated and correct value.

What is Model Fitting?

Model fitting is a measure of how well a machine learning **model** generalizes to similar data to that on which it was trained. A **model** that is well-**fitted** produces more accurate outcomes. A **model** that is overfitted matches the data too closely. A **model** that is underfitted doesn't match closely enough.

What is meant by epochs?

The beginning of a distinctive period in the history of anything, and the meaning here is to loop [one full iteration] over the full dataset.

What is the meaning of Flatten Layer?

Flattening is converting the data into a 1-dimensional array for inputting it to the next **layer**. We **flatten** the output of the convolutional **layers** to create a single long feature vector. And it is connected to the final classification model, which is called a fully-connected **layer**.

Ex: Convert 2D image into a Vector of feature.

What is meant by RELU?

ReLU is the most commonly **used** activation function in neural networks, It is used mainly to solve hard problems.

What is Softmax used for?

Softmax is often **used** in neural networks, to map the non-normalized output of a network to a probability distribution over predicted output classes.

What is Kernel Convolution ?

In image processing, a kernel, convolution matrix, or mask is a small matrix. It is used for blurring, sharpening, embossing, edge detection, and more.

In other words : applying a filter to an image is the kernel convolution.

What is MaxPooling ?

Max pooling is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned.

You should choose both pool size and stride.

New image from applying Max-Pooling is smaller than the original image {downsample} for the image.

What is Batch Size ?

Number of training examples to process before updating our Model variables.

Note that : batch size is equal to the total dataset thus making the iteration and epoch values equivalent.

What is Data Augmentation?

Data Augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset. ... The Keras **deep learning** neural network library provides the capability to fit models using image **data augmentation** via the ImageDataGenerator class.

What is ImageNet project?

ImageNet project is a large visual database designed for use in visual object recognition software research. More than 14 million images have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images.

What is Transfer Learning?

Transfer learning is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks.

In other Words: Use a pre-trained model with some modification instead of a new model.


What is Image Padding?

Image padding introduces new pixels around the edges of an **image**.

The border provides space for annotations or acts as a boundary when using advanced filtering techniques

What is Binary Cross-Entropy Loss?

Binary Cross-Entropy Loss Also called Sigmoid **Cross-Entropy** loss. It is a Sigmoid activation plus a **Cross-Entropy** loss. ... It's called **Binary**



Cross-Entropy Loss because it sets up a **binary** classification problem between $C' = 2$ classes for every class in C .

For More Info:

<https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a>

For More Info about different loss functions:

https://gombru.github.io/2018/05/23/cross_entropy_loss/

Why Data Generator: If you want to load a dataset but there is **not enough memory** in your machine. As the field of machine learning progresses, this problem becomes more and more common. Today this is already one of the challenges in the field of vision where large datasets of images and video files are processed. **{Dynamic Loading}**

For More Info:

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

Part 2: Group Report:

MRNET Model

The primary building block of MRNet is a convolutional neural network (CNN), mapping a 3-dimensional MRI series to a probability:

```
# 1st Convolutional Layer
ALEXNet.add(Conv2D(filters=96, input_shape=(227,227,3), kernel_size=(11,11), strides=(4,4), padding='valid'))
ALEXNet.add(Activation('relu'))

# Max Pooling
ALEXNet.add(MaxPooling2D(pool_size=(3,3), strides=(2,2), padding='valid'))

# 2nd Convolutional Layer
ALEXNet.add(Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), padding='valid'))
ALEXNet.add(Activation('relu'))

# Max Pooling
ALEXNet.add(MaxPooling2D(pool_size=(3,3), strides=(2,2), padding='valid'))

# 3rd Convolutional Layer
ALEXNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='valid'))
ALEXNet.add(Activation('relu'))

# 4th Convolutional Layer
ALEXNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='valid'))
ALEXNet.add(Activation('relu'))

# 5th Convolutional Layer
ALEXNet.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='valid'))
ALEXNet.add(Activation('relu'))

# To build avg_pooling cnn layers.
average_pool = Sequential()
average_pool.add(layers.AveragePooling2D())
average_pool.add(layers.Flatten())
# To use alex net as feature extractor.
#average_pool.add(layers.Dense(1, activation='sigmoid'))

# Bild MRNET.
MRNet = Sequential([
    ALEXNet,
    average_pool])

# Maxpooling
MRNet.add(Dense(256, activation='relu', kernel_constraint=keras.constraints.MaxNorm(max_value=2, axis=0)))
MRNet.add(Dense(1, activation='sigmoid'))
# stochastic gradient descent
sgd = optimizers.SGD(lr=1e-4, decay=1e-6, momentum=0.9, nesterov=True)

ALEXNet.summary()
average_pool.summary()
MRNet.summary()
```

AlexNet architecture consists of five convolutions and three fully-connected layers. To build the MRNET model, We used an average pooling CNN layer after the AlexNet layer to reduce these features to $s \times 256$ at the end we use Max-Pooling to across slices to obtain a 256-dimensional vector, which is passed to a fully connected layer to obtain a prediction probability.

In Fig 2: the The MRNet is a convolutional neural network (CNN) that takes as input a series of MRI images and outputs a classification prediction. AlexNet features from each slice of the MRI series are combined using a max pooling (element-wise maximum) operation. The resulting vector is fed through a fully connected layer to produce a single output probability. We trained a different MRNet for each task (abnormality, anterior cruciate ligament [ACL] tear, meniscal tear) and series type (sagittal, coronal, axial), resulting in 9 different MRNets (for external validation, we use only the sagittal plane ACL tear MRNet). For each model, the output probability represents the probability that the model assigns to the series for the presence of the diagnosis.

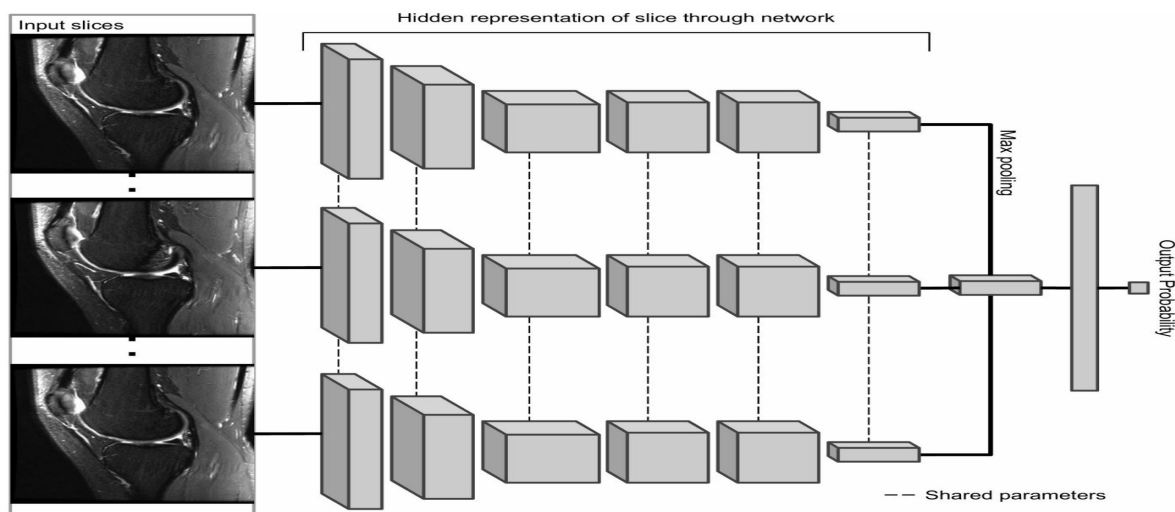


Fig 2. MRNet architecture. The MRNet is a convolutional neural network (CNN) that takes as input a series of MRI images and outputs a classification prediction. AlexNet features from each slice of the MRI series are combined using a max pooling (element-wise maximum) operation. The resulting vector is fed through a fully connected layer to produce a single output probability. We trained a different MRNet for each task (abnormality, anterior cruciate ligament [ACL] tear, meniscal tear) and series type (sagittal, coronal, axial), resulting in 9 different MRNets (for external validation, we use only the sagittal plane ACL tear MRNet). For each model, the output probability represents the probability that the model assigns to the series for the presence of the diagnosis.

Building Well-Known Networks Models for Feature Extraction

We are required to build Visual Geometry Group (VGG), Residual neural network (ResNet), Inception V3 Models:

VGGModel:


This is the code implementation with the structure of the model:

```

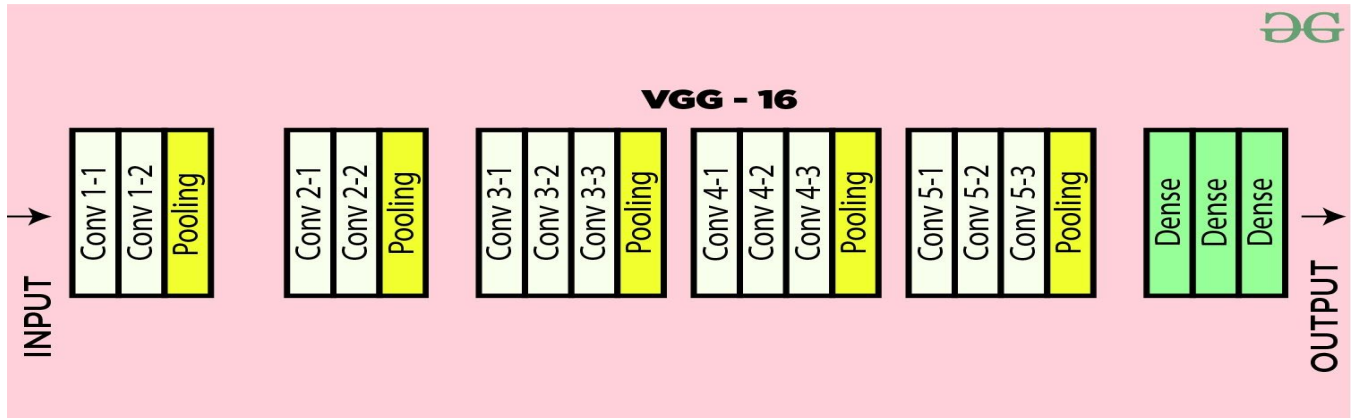
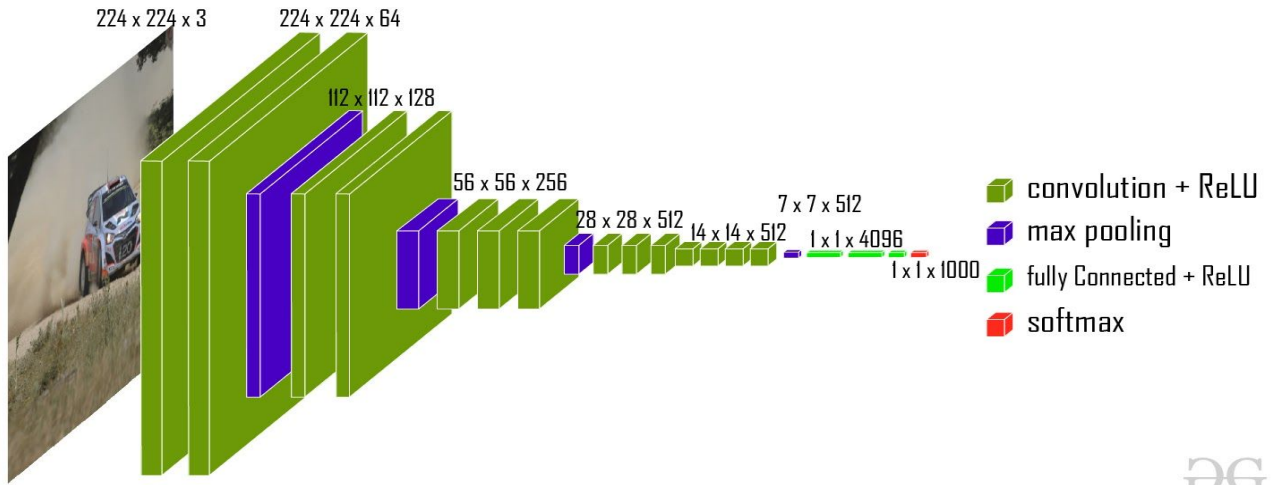
▶ vggModel = Sequential()
  vggModel.add(Conv2D(input_shape=(227,227,3),filters=64,kernel_size=(3,3),padding="same", activation="relu"))
  vggModel.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
  vggModel.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
  vggModel.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
  vggModel.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
  vggModel.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
  vggModel.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
  vggModel.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
  vggModel.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
  vggModel.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
  vggModel.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
  vggModel.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
  vggModel.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
  vggModel.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
  vggModel.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
  vggModel.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
  vggModel.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
  vggModel.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
  vggModel.add(Flatten())
  vggModel.add(Dense(units=4096,activation="relu"))
  vggModel.add(Dense(units=4096,activation="relu"))
  vggModel.add(Dense(units=2, activation="softmax"))
  from keras.optimizers import Adam
  opt = Adam(lr=0.001)
  vggModel.compile(optimizer=opt, loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])

  vggModel.summary()

```


 Model: "sequential_1"

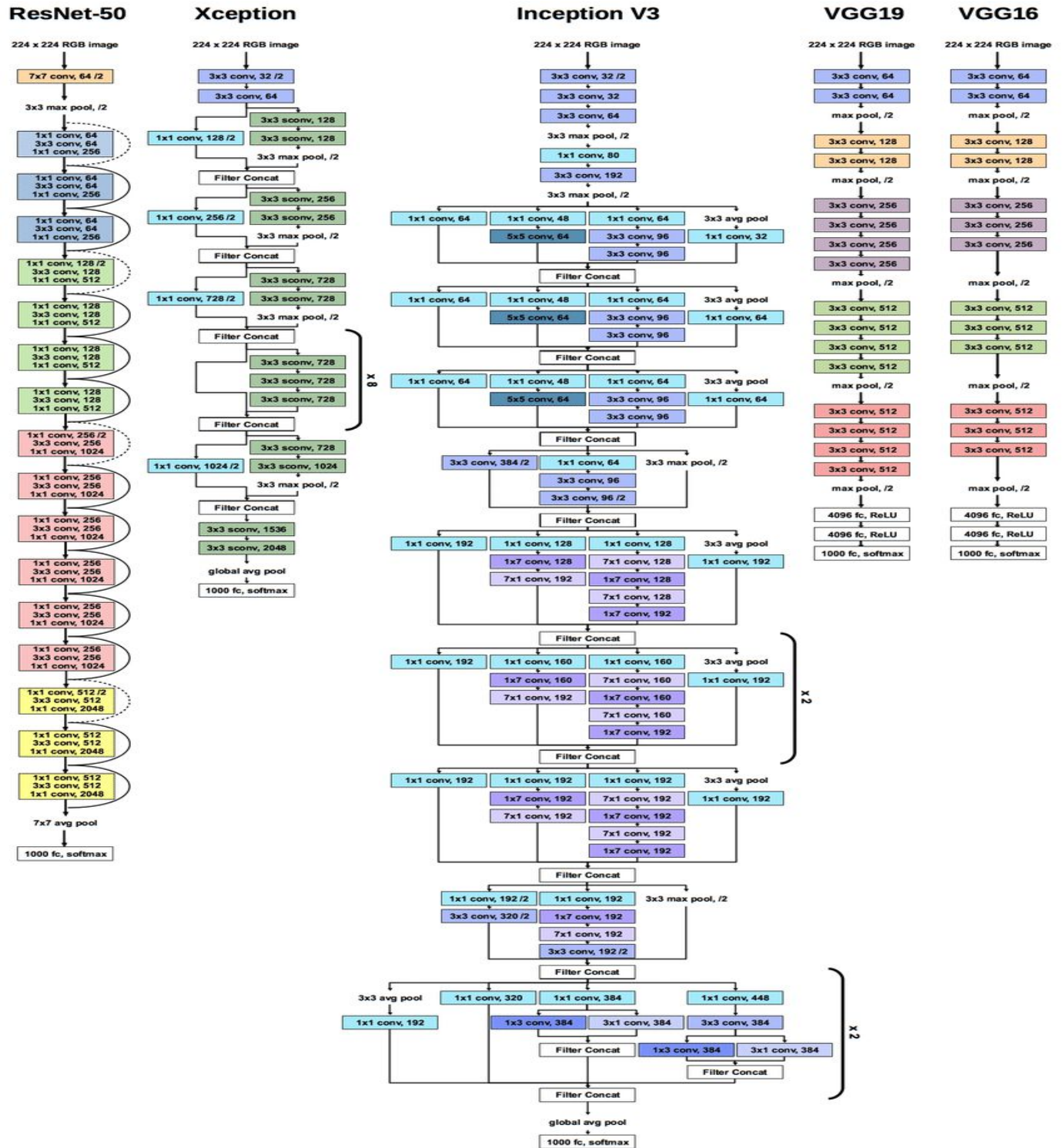
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 224, 224, 64)	1792
conv2d_2 (Conv2D)	(None, 224, 224, 64)	36928
max_pooling2d_1 (MaxPooling2D)	(None, 112, 112, 64)	0
conv2d_3 (Conv2D)	(None, 112, 112, 128)	73856
conv2d_4 (Conv2D)	(None, 112, 112, 128)	147584
max_pooling2d_2 (MaxPooling2D)	(None, 56, 56, 128)	0
conv2d_5 (Conv2D)	(None, 56, 56, 256)	295168
conv2d_6 (Conv2D)	(None, 56, 56, 256)	590080
conv2d_7 (Conv2D)	(None, 56, 56, 256)	590080
max_pooling2d_3 (MaxPooling2D)	(None, 28, 28, 256)	0
conv2d_8 (Conv2D)	(None, 28, 28, 512)	1180160
conv2d_9 (Conv2D)	(None, 28, 28, 512)	2359808
conv2d_10 (Conv2D)	(None, 28, 28, 512)	2359808
max_pooling2d_4 (MaxPooling2D)	(None, 14, 14, 512)	0
conv2d_11 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_12 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_13 (Conv2D)	(None, 14, 14, 512)	2359808
max_pooling2d_5 (MaxPooling2D)	(None, 7, 7, 512)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 4096)	102764544
dense_2 (Dense)	(None, 4096)	16781312
dense_3 (Dense)	(None, 2)	8194
=====		
Total params: 134,268,738		
Trainable params: 134,268,738		
Non-trainable params: 0		

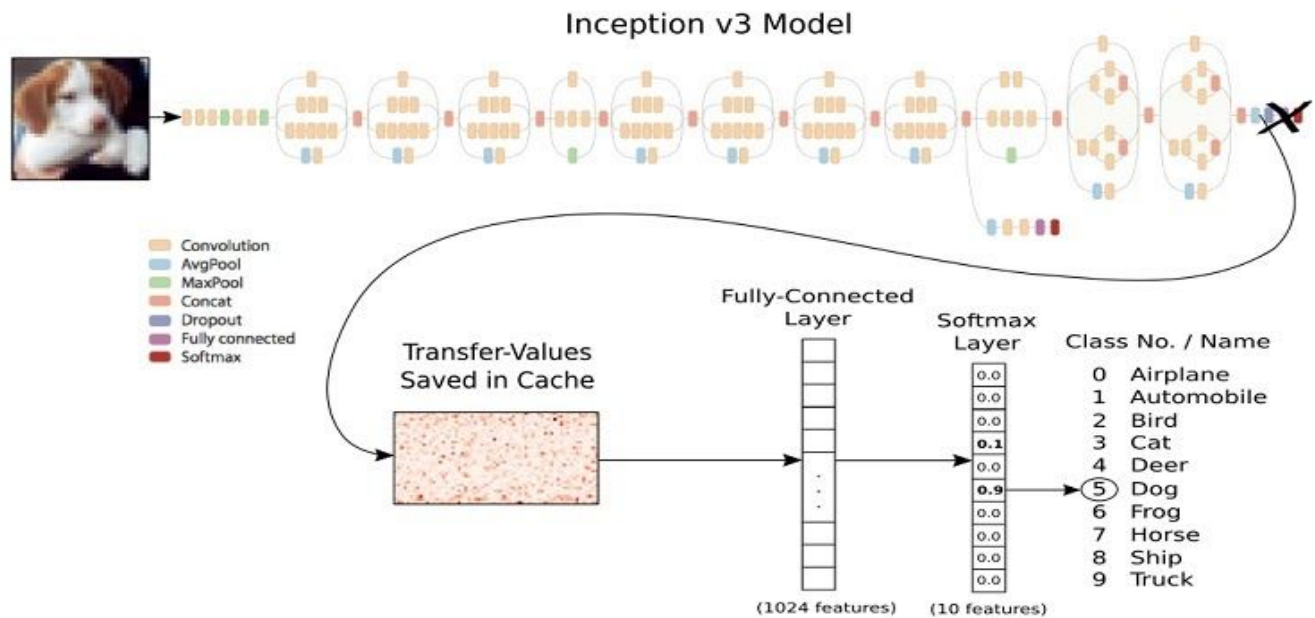


ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

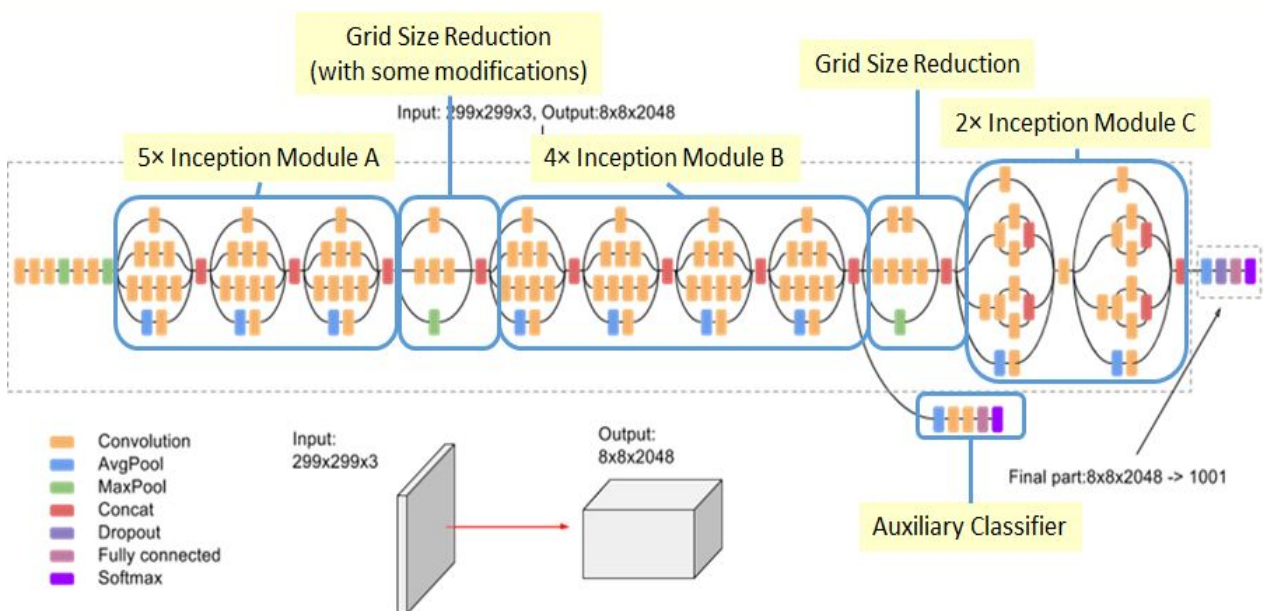
InceptionV3:

Here is the structure of the Inception V3 Model:



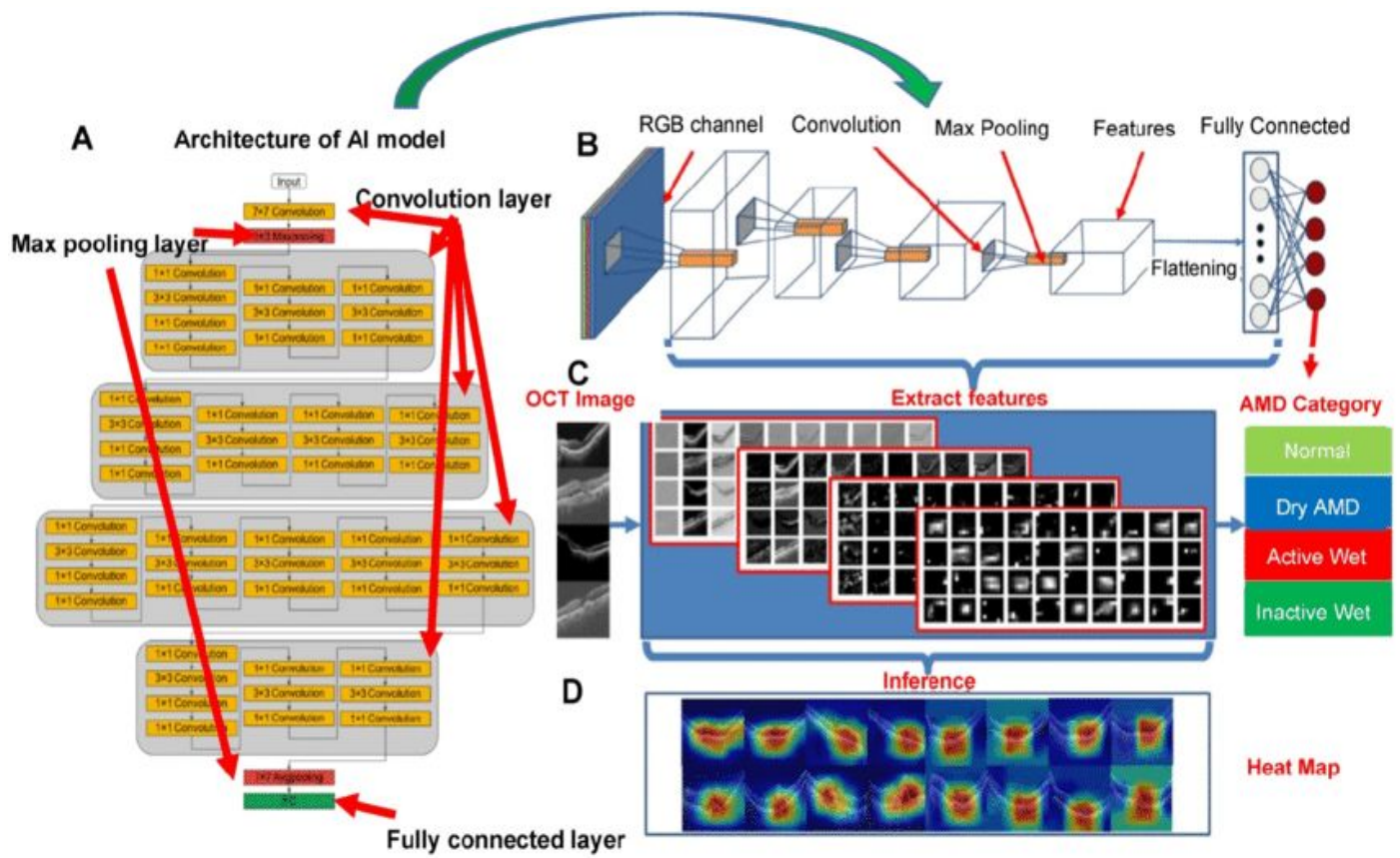


Note that: the input image format for this model is different than for the VGG16 and ResNet models (299x299 instead of 224x224), and that the input preprocessing function is also different (same as Xception).

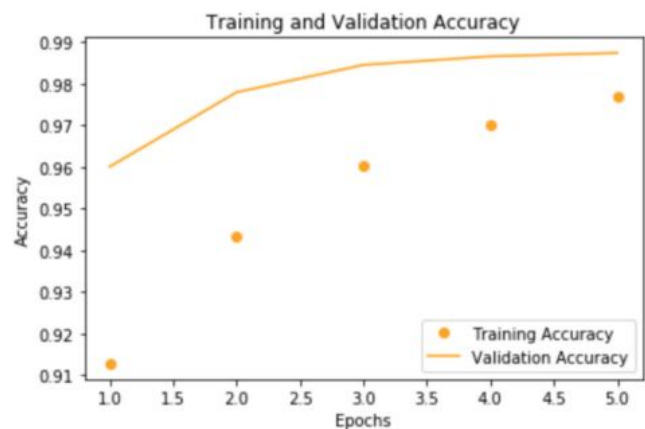
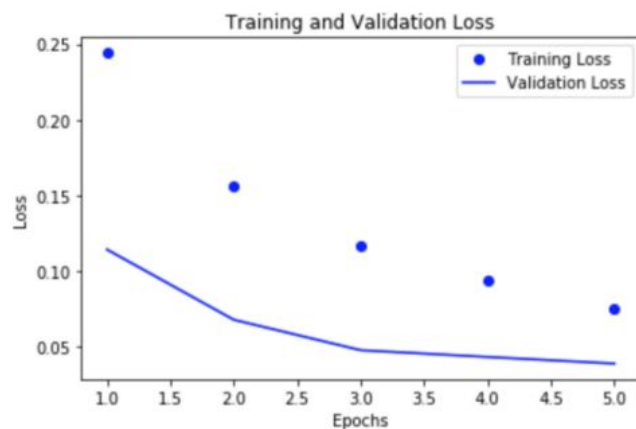


ResNet50:

Here is the model and the structure for the ResNet50:

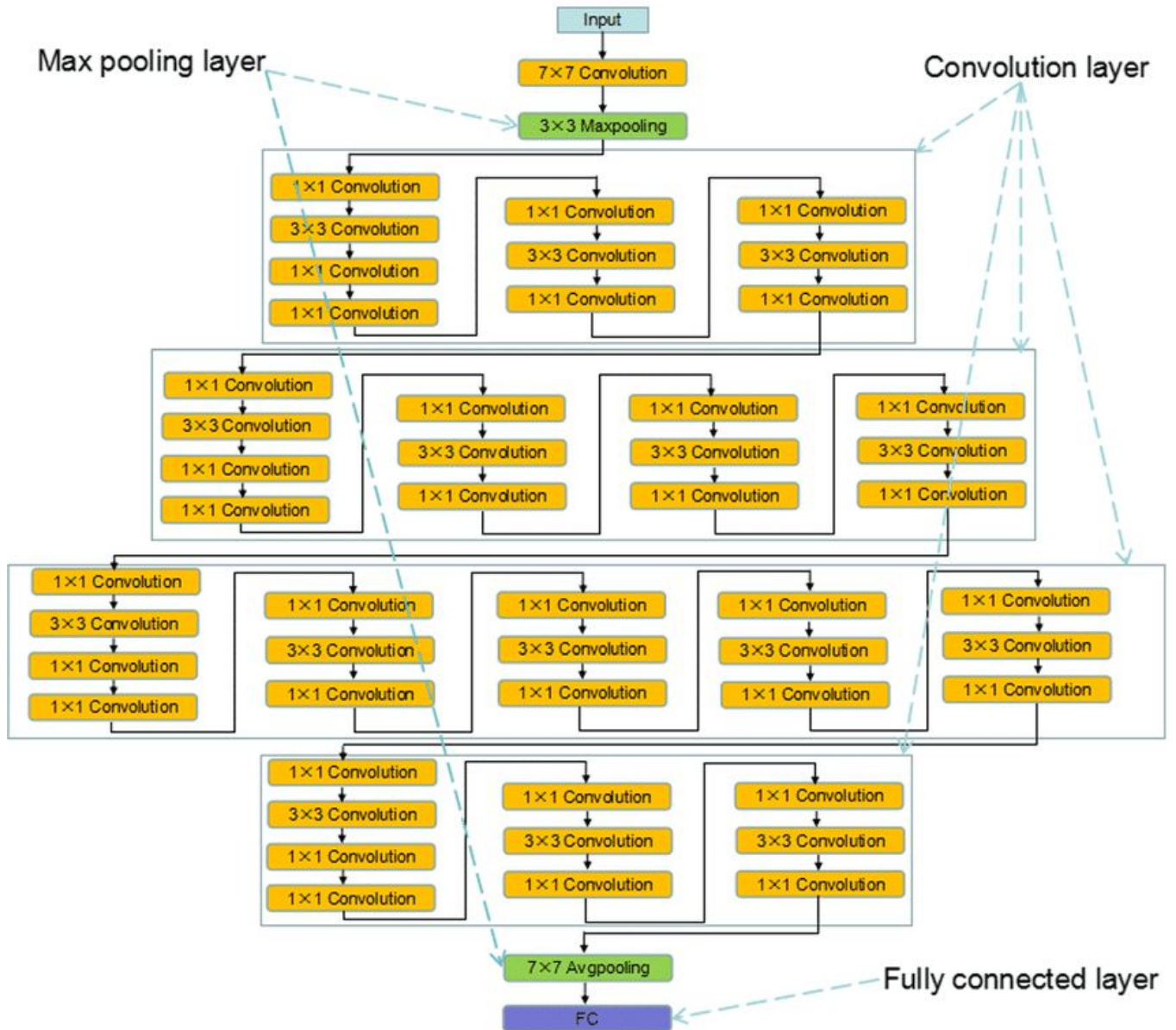


RestNet Validation and Accuracy Comparison.



a

Architecture of ResNet50 model



Transfer Learning:

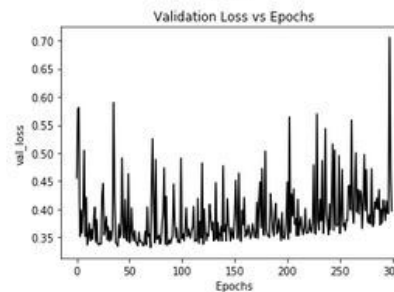
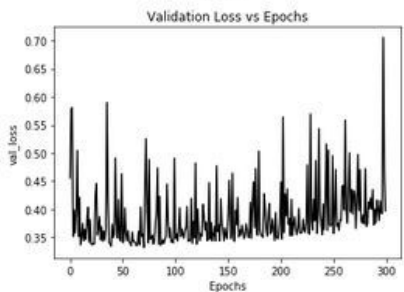
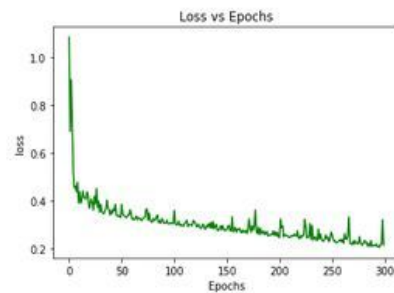
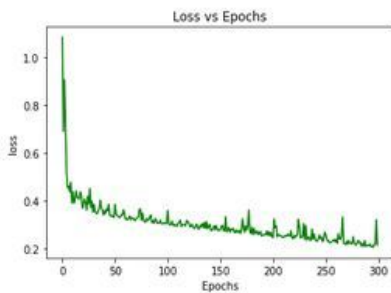
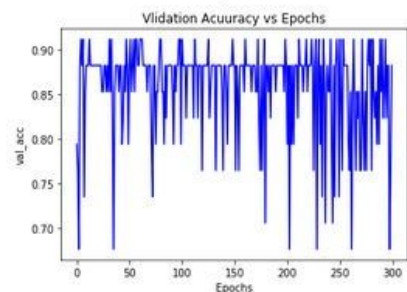
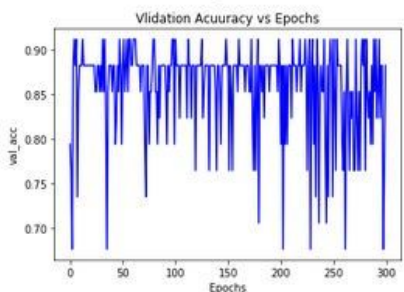
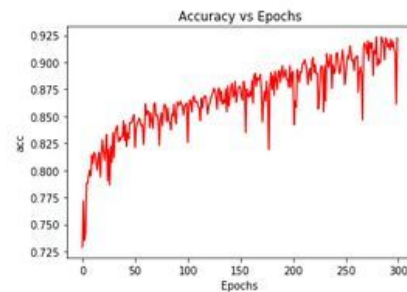
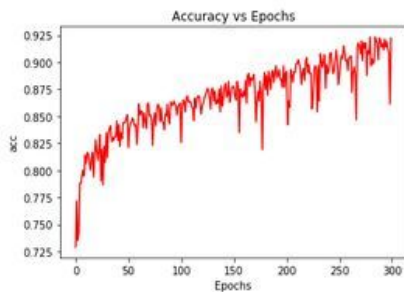
We use ResNet to do the transfer learning with image net weights.

Our Accuracy to predict :

1- Abnormal : 85.08496701055103 %

2- ACL : 55.398693366183174 %

3- Meniscus : 81.12091491619746 %



Steps:

Step1: Download & Understand data:

We downloaded the dataset from the Stanford site and it contained the following:

6 CSV files , 3 for train and 3 for validation, containing the labels of the 3 different sicknesses in a single picture of a knee. 1 csv file for abnormal labels , 1 for acl tears and 1 for meniscus tears.


A train folder and a validate folder containing numpy files that contains the MRI of the knee in the form of slices , meaning that each numpy file has a particular number of slices representing the MRI of that specific knee for that specific view (Axial , Coronal , Sagittal) and it differs for each numpy file. However the pixels of each slice are the same [256,256] .

This concludes the step of downloading and understanding the dataset.

Step2: Pre-Processing:

We had to do some kind of pre-processing on the data after reading it. We started by reading the data from google drive where we uploaded it, using panda frames we read all 3 csv label files and put them in dataframes and then we created a function that loops on the train folder directory and reading the numpy files into the shape (s,256,256) where s is the number of slices for the particular MRI.

The first pre processing thing we had to do was unite the s number for all the images and we did that using a function that finds the picture with the minimal number of slices and that the number we go with for all other pictures, we made a function that loops on all slices of each picture and choose a random combination of that same minimal number of slices previously decided.



The second pre processing step we had to was turn the 3 dimensional shape into a 4 dimensional shape because as we find out later the ResNet50 model only takes 4 dimensional shapes, so we did that by turning the 1 channel image or [grey scale] into a 3 channel image [RGB], turning it into the dimension (s,256,256,3) we did that using the CV2 resize function.

The third and last pre processing step we did was for the labels, the labels as we mentioned above were given for each picture not for each slice, so seeing that we decided to do our training on slice base information we had to enlarge the label data to be the same size and concept of the training data , so we made a function that repeated the label of each picture n times, n being the minimum number of slices in that entire MRI folder view.

And that gave us our training labels.

This concludes the pre-processing part of our program.

Step3: Building The CNN Model:

This step is the implementation of our CNN Models, and we used AlexNet with Average pooling too to Implement MRNET Model.

We then fit to our training data and training labels for the abnormal labels and we evaluate() on the test data and test abnormal labels we save the output of the evaluate() which is the loss and accuracy.

This way we end up with 9 accuracies , 3 for each view, 1 for each labeled sickness.