Alexandria University
CSED21
Mohamed Mohamed Abdel Hakim Mohamed(45).

MRNet implementation with Inception V3

2nd June 2020

OVERVIEW

I have implemented the inception V3 feature extractor from its paper[1]. The model training includes lots of trials and the parameter tuning pushed the model to fit the MRI scan. This parameter tuning has been applied to the 9 models corresponding to each combination of exam angles and class type (abnormal, ACL, meniscus). All the 3 models corresponding to the class Type combined using a logistic regression model to weight the contribution of each model to the output prediction.

IMPLEMENTATION PIPELINE

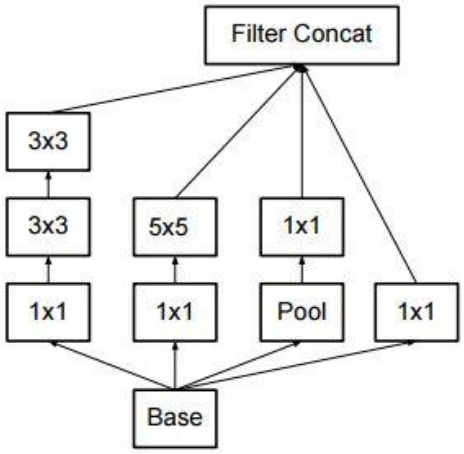
1. I have Implemented 5 basic CNN modules of inception V3, 3 inception modules of factorized convolutions (figure 5, 6, 7 in the paper), and 2 grid-size-reduction modules (figure 10 in the paper).
2. The second step is to then combine the modules to form a feature extractor Keras model with an output feature map (8x8x2048). Then, I have implemented a custom layer to call the feature extractor and combine the feature maps of the input batch and pass it to a sigmoid classifier.
3. I have added the custom layer to a sequential model then compiled the model and created a callback to save the weights of the functions.
4. Then I trained the models under different parameter settings that I will mention in the Trails section. After I tuned the model, I applied the same tuning to 9 different models then started training all of them.
5. After that, I used the validation set (10% of the training set) to evaluate the model against a different set of weights then I have chosen the best set of weights and tested the model using the external validation set
6. The last step is to combine predictions from the three different angles in the logistic regression model then train the model to adjust the weights of the different exam angles.

[1] Rethinking the inception architecture for computer vision

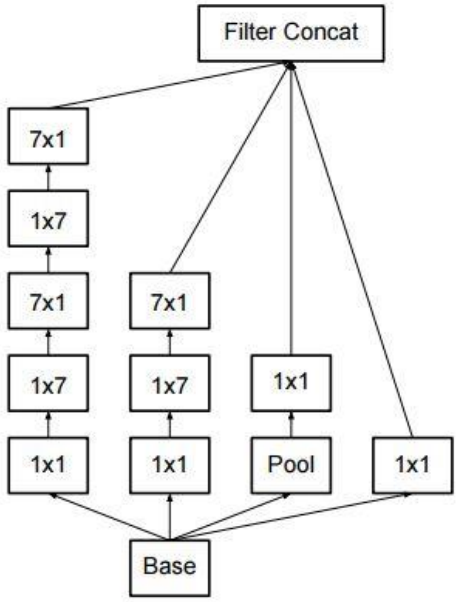
INCEPTION V3 Summary

Here, I am going to summary the key points of inception v3 architecture. And how I have implemented it, I have implemented the 5 main modules in inception. I have gained insights about the channel depth of the modules from intel article[2] of inception v3. The First keypoint in inception v3 is factorization of large convolutions with smaller convolutions which results in the same performance but with reduced computation. here are the modules in my implementation:

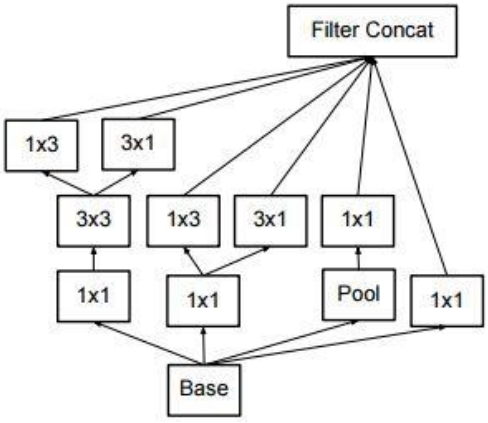
Module A:



Module B:

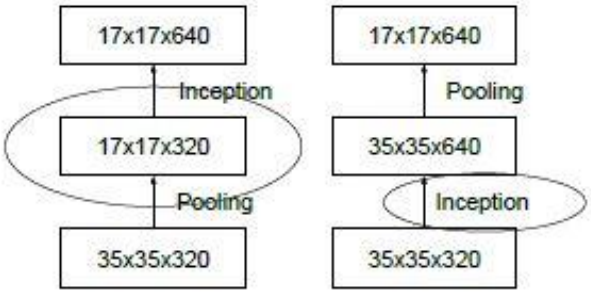


Module C:

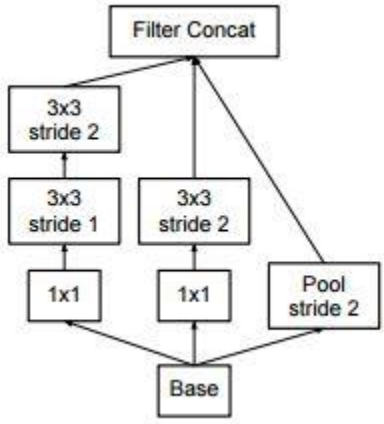


[2] https://software.intel.com/content/www/us/en/develop/articles/inception-v3-deep-convolutional-architecture-for-classifying-acute-myeloidlymphoblastic.html

Efficient Grid size reduction: this is used instead of the two alternative ways of reducing the grid size. The one on left violates the design principle of avoiding representational bottlenecks and the one on right is 3 times more expensive computationally.



Grid size-reduction module:



Module D: this module is a variation from the grid size-reduction module but without the bottleneck (1x1) convolution in the middle.

Module E: also it is a variation from grid size reduction but factorized the (3x3 stride 1) with two convolutions (1x7) and (7x1).

TRAINING TRAILS

All the training trials were done on the axial angle and abnormal class, then the best setting was generalized

1) First, I have tried training on the skewed data the results were not satisfactory. The model has not predicted any negative class, so the true negative and false negative metrics were zeros.
2) After balancing the dataset classes using oversampling and augmentation, I have applied the parameter settings (RMSprop optimizer) in inception v3 paper but also the model was not learning, the accuracy was still decreasing after 10 epochs.

3) During the implementation of inception, we added the auxiliary classifier by a walkthrough and duplicating the output label but the effect of this was not satisfactory, the model was not learning as the implementation of the auxiliary classifier was not appropriate, After reading more about the auxiliary classifier from inception v1 paper[3] we have found that its gradient weights are applied with 0.3 and added to the original gradients so tried removing the auxiliary classifier and started training again the model, the model training accuracy started to increase at a slow rate.
4) When applying batch norm after the convolution layers, google colab always goes with exhausted resources (tensor allocating), so I tried a different setting.
5) After training the network with different learning rates, we have found that what made the model stuck with an accuracy of about (50%-55%) is the vanishing gradients as the weights are slightly updated at each iteration.
6) So I applied the caviar approach and started training different models under different settings
    - Adam optimizer with learning rates =0.001, 0.0001, and data augmentation.the model accuracy also stuck after 10 epochs
    - Adadelta optimizer with learning rate =0.001 and data augmentation as a previous model. The model is learning but with a very slow rate after 10 epochs accuracy approximately 52%
    - Adadelta optimizer with learning rate = 0.0001 and stop augmentation to apply batch normalization. (succeeded)
    - RMSprop with learning rate = 0.045 (as in inception v3 paper).the accuracy oscillates around 49%.
7) applying batch normalization after each convolution has dramatically pushed the model training forward while maintaining the generalized performance on internal validation dataset, but no augmentation is applied at this model as when batch size increase as a result of augmentation, the computations grows significantly and colab goes into exhausted resources so retrain the model initialized with the previous training weights.
8) When evaluating the model using the training data, the 9 models produce accuracies within range (61%-76%) and the model predicts always the major class so I tried different training settings as may the models still under fits the data.
9) Also, I applied the caviar approach and trained different models in parallel:
    a) using dropout 0.1 instead of 0.5 speeds up training but testing no more improvement
    b) using different range of learning rates 1 ,0.1 ,0.001, 0.00001, 0.0001
    c) trying to overfit a small dataset and testing using the same dataset, the model evaluation accuracy no more improved
    d) apply non-trainable batch normalization no more improvement same results
    e) applying L2 regularization for kernels
    f) using adam optimizer instead of adadelta the accuracy decreases.
10) when trying to predict the output, the model predicts the same prediction for all outputs which is the majority class so this may happen because the model stuck in a local minimum and under fits the data.

[3] Going deeper with convolutions

11) training meniscus axial model for 50 epochs under learning rate 0.00001, the model overcomes the same class problem but still very small accuracy so more epochs may help and I load the same weights for the remaining 9 models. and train for 50 more epochs but the problem persists after the 50 epochs.
12) I tried another method to solve the skewed data and stopped oversampling and used class weights in Keras to balance the loss.
13) Also, I have implemented inception using the built-in Keras model to compare its performance with my implementation, the same class prediction problem persists with the imagenet weights of the built-in model.
14) after researching through problem space there are some introduced solutions:
    a) skip connections that take us to another CNN architecture (residual inception) [4]which is implemented in (inception v4)
    b) Remove all regularizers as dropout, L2 regularizer totally from the model, and apply label smoothing as a regularizer (inception v3 paper))with a small learning rate 1e-5. This trail has succeeded, and the model overcomes the same class prediction. The regularizer effect has made the model underfit the data.
15) Then I applied these settings to all the 9 models and trained them for 50-100 epochs.
16) I chose the best set of weights and applied them to predict the output probabilities using a variation from the data generator, we used in training, to has a sorted version of exams by their IDs and saved the predicted probabilities in CSV files and used them in training the logistic regression (LR) model to combine the probabilities with appropriate weights and train the LR model for 100 epochs.

[4] https://ieeexplore.ieee.org/document/8451515

EVALUATION RESULTS

The results of the 9 models

		Training		Validation	
		Accuracy	AUC	Accuracy	AUC
Abnormal	Axial	90.50%	94.73%	87.06%	87.75%
	Coronal	94.71%	97.81%	78.46%	69.09%
	Sagittal	92.68%	97.07%	85.48%	81.91%
ACL	Axial	97.96%	98.91%	74.41%	83.48%
	Coronal	88.38%	95.13%	78.81%	75.53%
	Sagittal	94.05%	97.64%	85.00%	74.01%
Meniscus	Axial	89.30%	94.26%	62.81%	61.43%
	Coronal	95.54%	97.71%	74.21%	73.69%
	Sagittal	71.04%	81.40%	70.83%	79.24%

The results after combining models

	Training accuracy	Training AUC	Validation Accuracy	Validation AUC
Abnormal	90.08%	95.29%	76.01%	86.75%
ACL	98.53%	99.55%	79.37%	61.77%
Meniscus	94.88%	99.03%	75.88%	61.84%