Shahid Beheshti University

Computer science

Master of science

Networks with Parallel Concatenations(GoogLeNet)

Mohammad Javad Aghaie
Tayebeh Taheri

Dr. Seyed Ali Katanforoush

Artificial Neural Networks

1400-1401

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., … Rabinovich, A. (2015). Going deeper with convolutions. Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1–9)

https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf

PROBLEM

Problem

- **Waste Classification**
  - ✓ Automatic recognition and detection of waste from images

origin of problem

- main method of managing the waste/garbage are:
  1. landfilling
  2. burning waste

- issues like:
  - ✓ inefficient
  - ✓ expensive
  - ✓ Consumption of toxic waste by animals
  - ✓ Leachate
  - ✓ Increase in toxins
  - ✓ Land, water and air pollution

Separating waste

- manual separation of objects by humans
- issues like:
  - ✓ Inefficient
  - ✓ Expensive
  - ✓ time consuming
  - ✓ prone to errors caused by limited knowledge of waste classification

Separating waste

- intelligent waste material classification system
- Pros:
  - ✓ take short time to sort the waste
  - ✓ more accurate in sorting than the manual way.

Applications

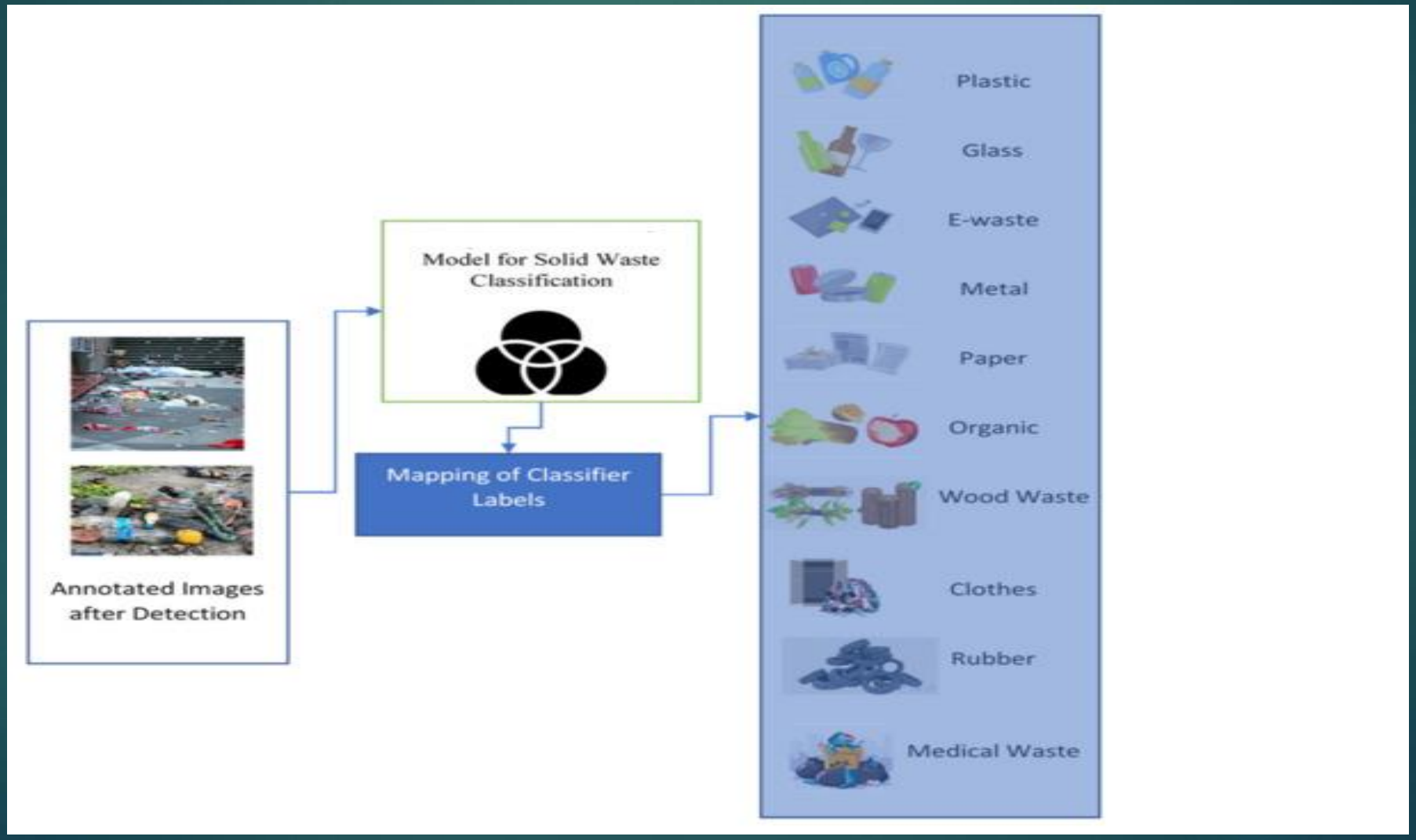
- **Waste Classification is the first and most important step for**
  1. Recycling waste
  2. Reusing materials such as glass, metal, paper, and electronics
  3. Reducing soil additives and methane

Problem in general

- Waste segregation using deep learning algorithm
  training, validation, and testing
- a given image can be annotated with an appropriate label
  identification & **classification** of waste

- **Input** data: trash image dataset
- **Output**: automatically classify waste into the relevant categories
  images categorized into a number of waste classes:
    glass, paper, cardboard, plastic, metal, and so on.

Problem in general



Our problem

Input: taking images of solid waste

Among the publicly available waste classification datasets is **Sekar's**

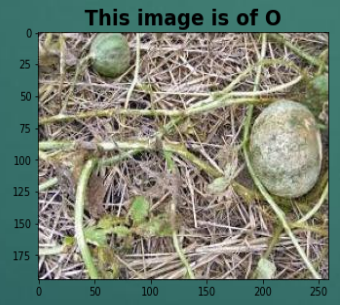
Output: Separating waste into two Classes:

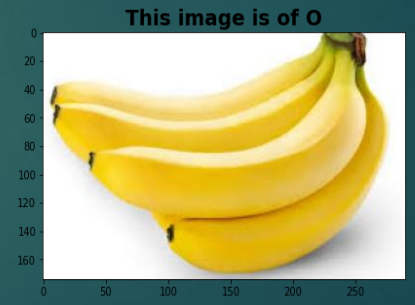
Organic: biodegradable

Recyclable: reusable materials such as glass, metal, paper, & electronics

O ⟶ Organic

R ⟶ Recyclable





MODEL

GoogleNet Model
Introduction

- GoogLeNet is a deep convolutional neural network
- was proposed by research at Google in 2014
- Won image classification challenge
- The **ImageNet** project is a large visual database designed:
- computer vision tasks such as **face detection** and **recognition**, **adversarial training**

Why GoogleNet?

- **Large networks are prone to**
- ✓ overfitting
- ✓ exploding
- ✓ vanishing gradient problem
- GoogLeNet reduce the **input image**, while retaining important information
- significantly **less error rate** than **VGG** and **AlexNet**
- has **10 times fewer parameters** than **AlexNet** (roughly 6 million instead of 60 million)
- It has **top-5** error rate of **6.67%** in classification task

Deal with overfitting and model performance improvement

- To overcome overfitting, there are different methods. Following we mention some of them:
  - **Ealy Stopping Point** : This helps training phase to stop as soon as before. As a result, it is unlikely to overfit the model.
  - **Regularization**: Like machine learning, in Neural Networks regularization term like L2 can help to deal with overfitting and increase model performance.
    - kernel_regularizer: Regularizer to apply a penalty on the layer's kernel
    - bias_regularizer: Regularizer to apply a penalty on the layer's bias
    - activity_regularizer: Regularizer to apply a penalty on the layer's output
  - **Dropout:** The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged.
- To improve performance, there are different methods. Following we mention some of them:
  - **Reduce Learning Rate On Plateau** : This strategies result in diminishment and eventually vanishment of learning rate. This works when we are near a local optimum (peak).
  - **Higher Learning rate and lower momentum**: this cause to reduce updating part of gradient so we need more epochs to reach a peak.
  - We have used some of these methods as it was needed

Early stopping

- Early stopping is a famous method for preventing overfitting
- It stops model as soon as the criteria has reached
- “Patience” : Number of epochs with no improvement after which training will be stopped.
- “restore_best_weights” :Whether to restore model weights from the epoch with the best value of the monitored quantity. If False, the model weights obtained at the last step of training are used. An epoch will be restored regardless of the performance relative to the baseline. If no epoch improves on baseline, training will run for patience epochs and restore weights from the best epoch in that set.

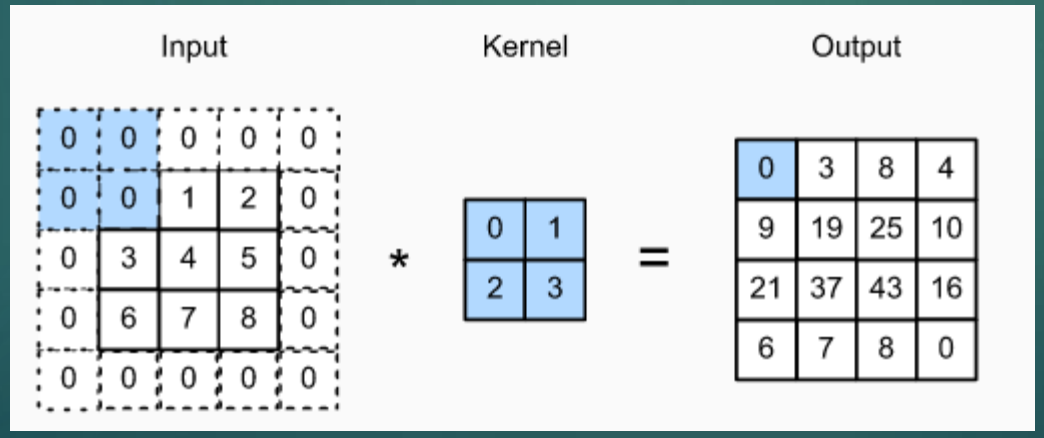
Introduction to GoogleNet:

- using blocks called an Inception block
- using combination of variously-sized kernels
- Inspired of VGG and NiN models:

padding

- adding extra pixels of filler around the boundary of input image
- preserves the size of the original image
- $n*n$ matrix convolved with an f*f matrix the with padding p=1

output image: (n + 2p — f + 1) * (n + 2p — f + 1)

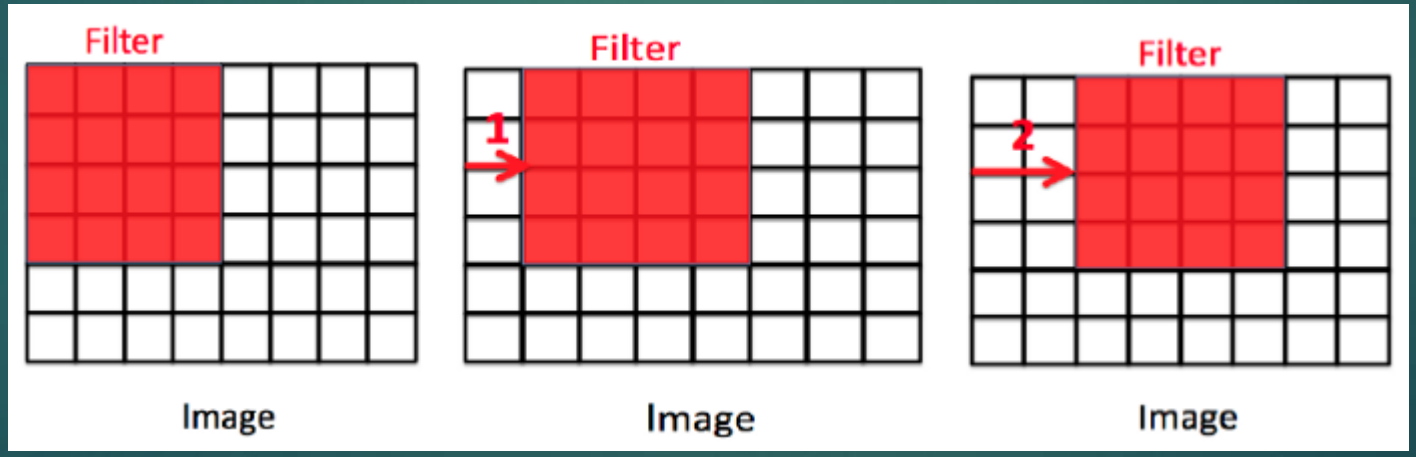


Stride

- is the number of pixels shifts over the input image
- For padding p, filter size $f*f$, input image size $n * n$ and stride 's'

output image dimension:

$[\{(n + 2p - f + 1) / s\} + 1] * [\{(n + 2p - f + 1) / s\} + 1]$



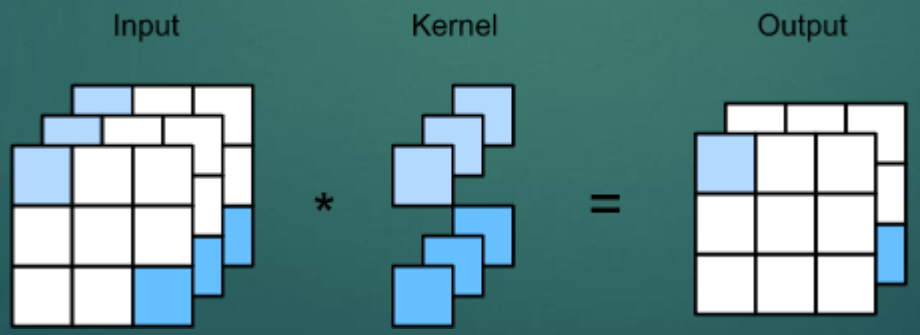
left image: stride =0, middle image: stride = 1, right image: stride =2

Features of GoogleNet:

- **1×1 convolution**
- **Global average pooling**
- **Inception Blocks**

1×1 convolution

- using the 1×1 convolution
- inputs and outputs have the same height and width
- decrease the number of parameters (weights and biases)

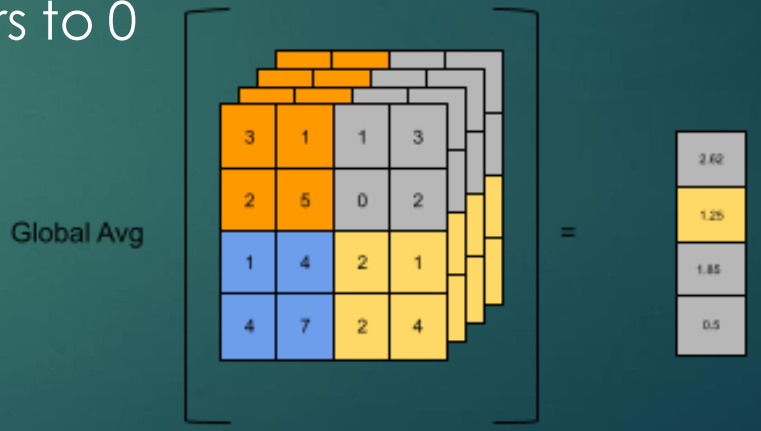


Global Average Pooling

- takes the average of each feature map
- generates **one feature map** in the **last Conv layer**
- takes a feature map of **7×7** and averages it to ***1×1***
- advantage:

✓ decreases the number of trainable parameters to 0

✓ overfitting is avoided at this layer



Inception Blocks

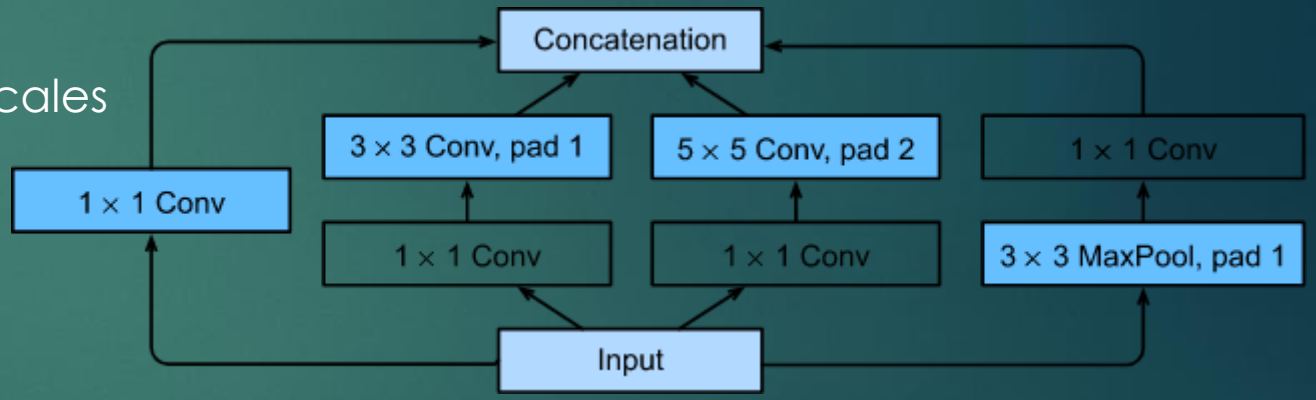
- **Inception block consists of four parallel paths**
- **ReLU activation function**
- **first three paths use convolutional layers:**

  Size: 1×1, 3×3, 5×5

  allowing them to capture patterns at different scales
- **Using 1×1 convolution on the input:**

  to reduce the number of channels
- **fourth path uses a 3×3 maximum pooling**
- **every single layer uses a stride of 1**

  outputs all have the same height and width as their inputs
- **outputs along each path are concatenated**
  - ✓ 3 – 10× faster than similarly performing networks with non-Inception architecture

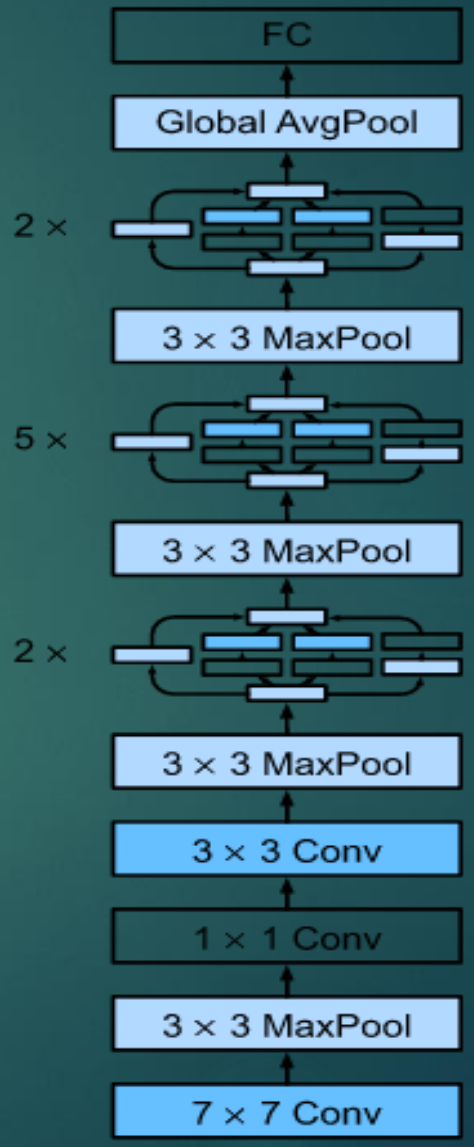


GoogleNet Model

An Introduction

GoogLeNet Model

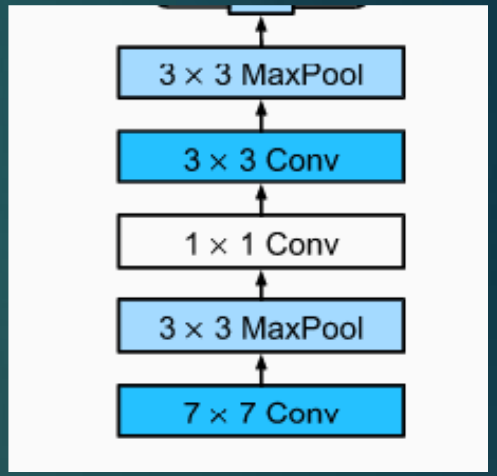
- GoogleNet uses:
- 9 inception blocks
- consists of 22 layers
- global average pooling
- rectified linear activation
- **Maximum pooling** (27 layers if we count pooling)
- first module is similar to AlexNet
- blocks is inherited from VGG



Architectural Details:

- first module uses a 64-channel 7×7 convolutional layer, stride 2

Purpose: immediately reduce the input image



- The second module uses two convolutional layers:

64-channel 1×1 convolutional layer

A 3×3 convolutional layer, stride2

corresponding to the second path in the Inception block

- 3×3 max-pooling, stride2

Third module

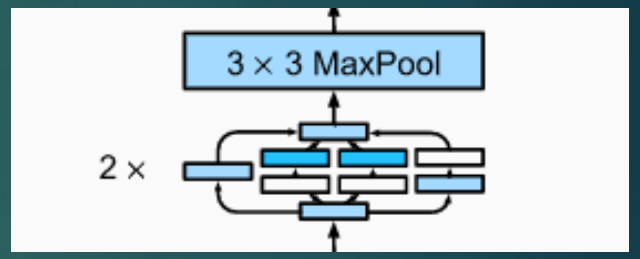
- third module connects **two** complete Inception blocks
- number of output channels:

1. First block: 64+128+32+32=256

Output size: 28×28×256

1. Second block: 128+192+96+64=480

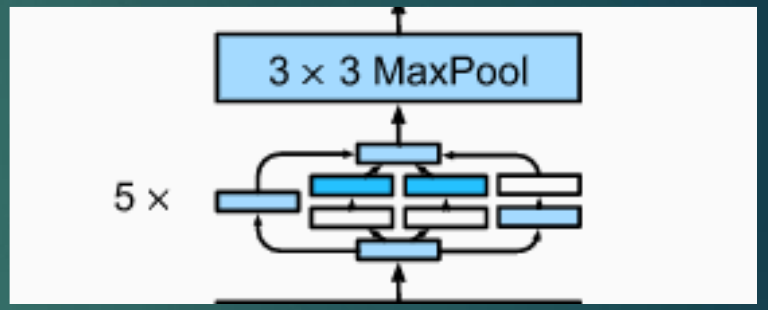
Output size: 28×28×480



fourth module

- five Inception blocks in series
- output channels:

1. 192+208+48+64=512, output size: 14×14×512
2. 160+224+64+64=512, output size: 14×14×512
3. 128+256+64+64=512, output size: 14×14×512
4. 112+288+64+64=528, output size: 14×14×528
5. 256+320+128+128=832, output size: 14×14×832

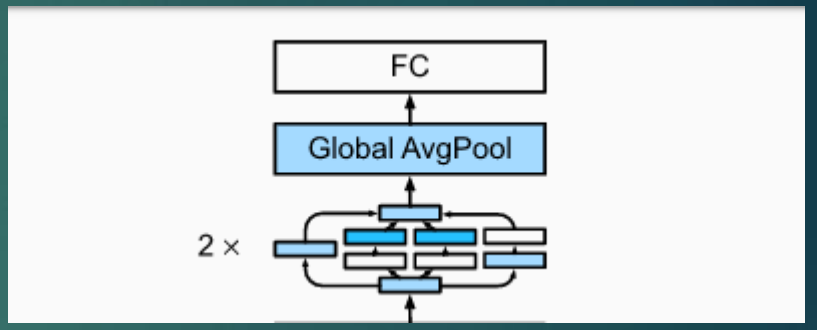


fifth module

- two Inception blocks
- Output channels:

1. 256+320+128+128=832
2. 384+384+128+128=1024

- fifth block is followed by the **output layer**
- global average pooling: height and width of each channel is 1
- fully-connected layer: output ➡ number of label classes



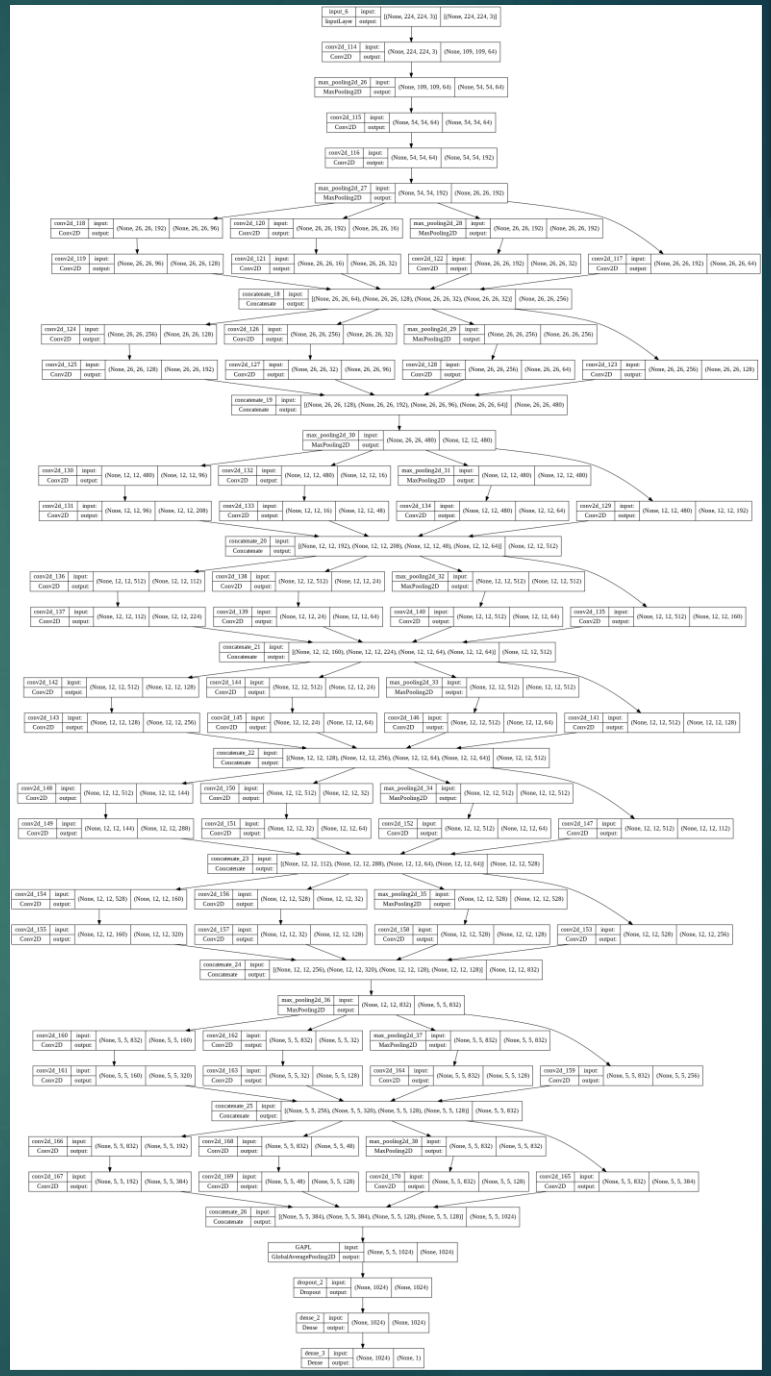
Output (prediction Layer)

- Since we have 2 classes, we needs a dense layer with one output.
- We chose “sigmoid” as Activation Function. This gives up the probability of each class. For showing output we need to have a threshold to determine which class the output belongs to.

overview

- purpose of max-pooling layers is to downsample the input
- Reducing the input size between the inception module is another effective method
- average pooling layer reduced the input height and width to 1x1
- The final layer uses the sigmoid function
- All the convolutional layers use the ReLU activation function

Our GoogleNet Model Structure



Inception V3

NEW VERSION OF GOOGLENET

Inception V3

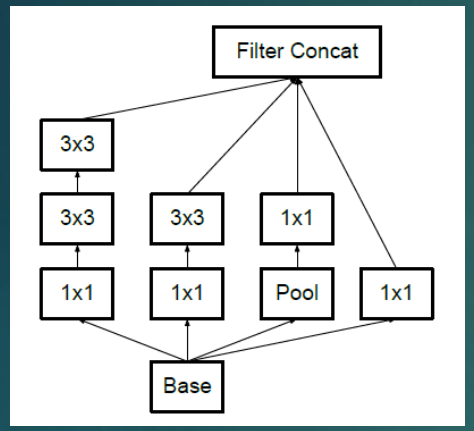
- Inception-v3 is a CNN architecture from the Inception family
- has a total of 42 layers
- The major modifications on the Inception V3 model are:
  1. **Factorization into Smaller Convolutions**
  2. **Spatial Factorization into Asymmetric Convolutions**
  3. **Utility of Auxiliary Classifiers**
  4. **Efficient Grid Size Reduction**:

     if we have a d×d grid with k filters after reduction it results in a d/2 × d/2 grid with 2k filters
  5. **using Label Smoothing**:

     regularizes a model based on a softmax with k output

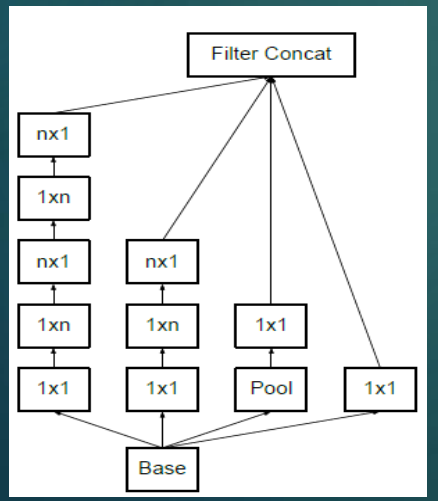
Inception V3

1. Factorization into Smaller Convolutions

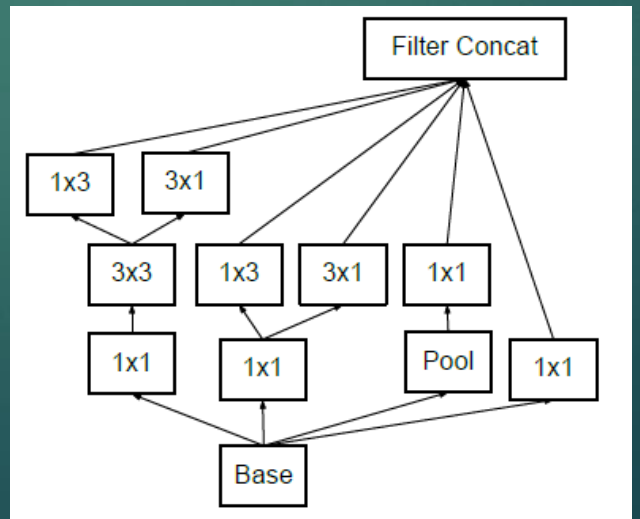


module 1

2. Spatial Factorization into Asymmetric Convolutions

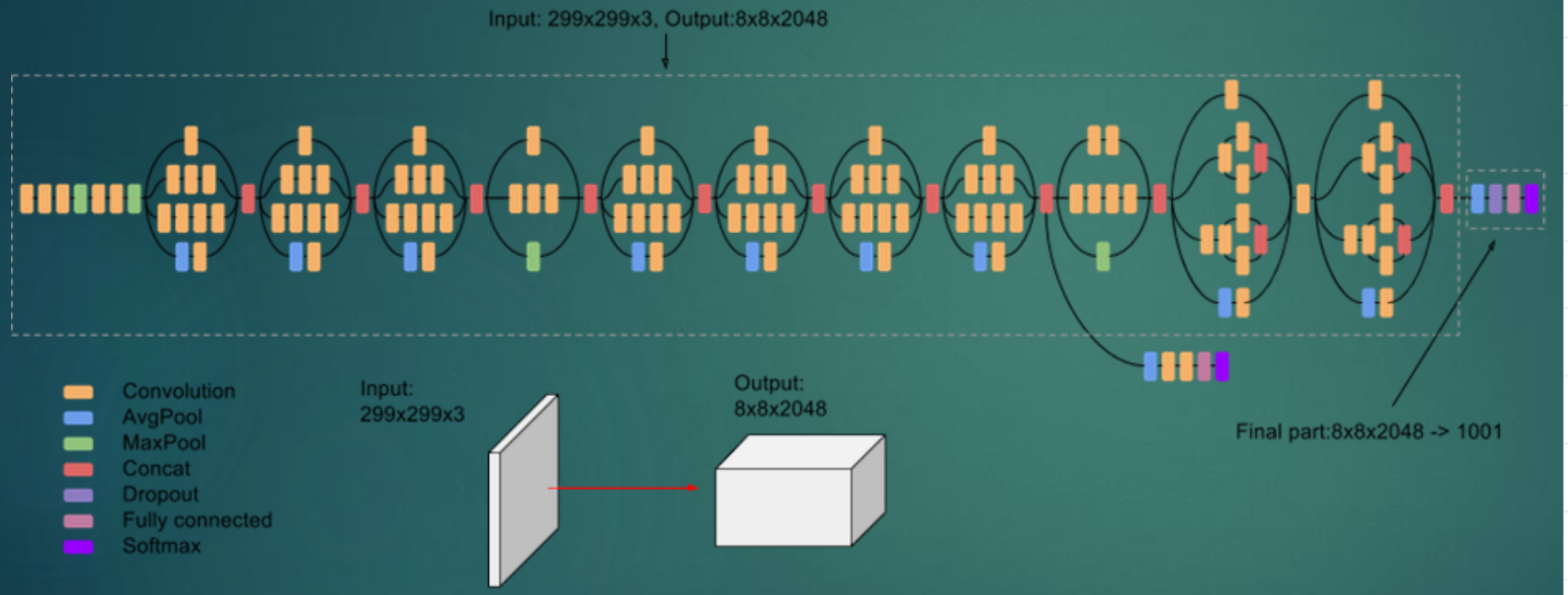


module 2



module 3

Inception V3



TYPE	PATCH / STRIDE SIZE	INPUT SIZE
Conv	3×3/2	299×299×3
Conv	3×3/1	149×149×32
Conv padded	3×3/1	147×147×32
Pool	3×3/2	147×147×64
Conv	3×3/1	73×73×64
Conv	3×3/2	71×71×80
Conv	3×3/1	35×35×192
3 × Inception	Module 1	35×35×288
5 × Inception	Module 2	17×17×768
2 × Inception	Module 3	8×8×1280
Pool	8 × 8	8 × 8 × 2048
Linear	Logits	1 × 1 × 2048
Softmax	Classifier	1 × 1 × 1000

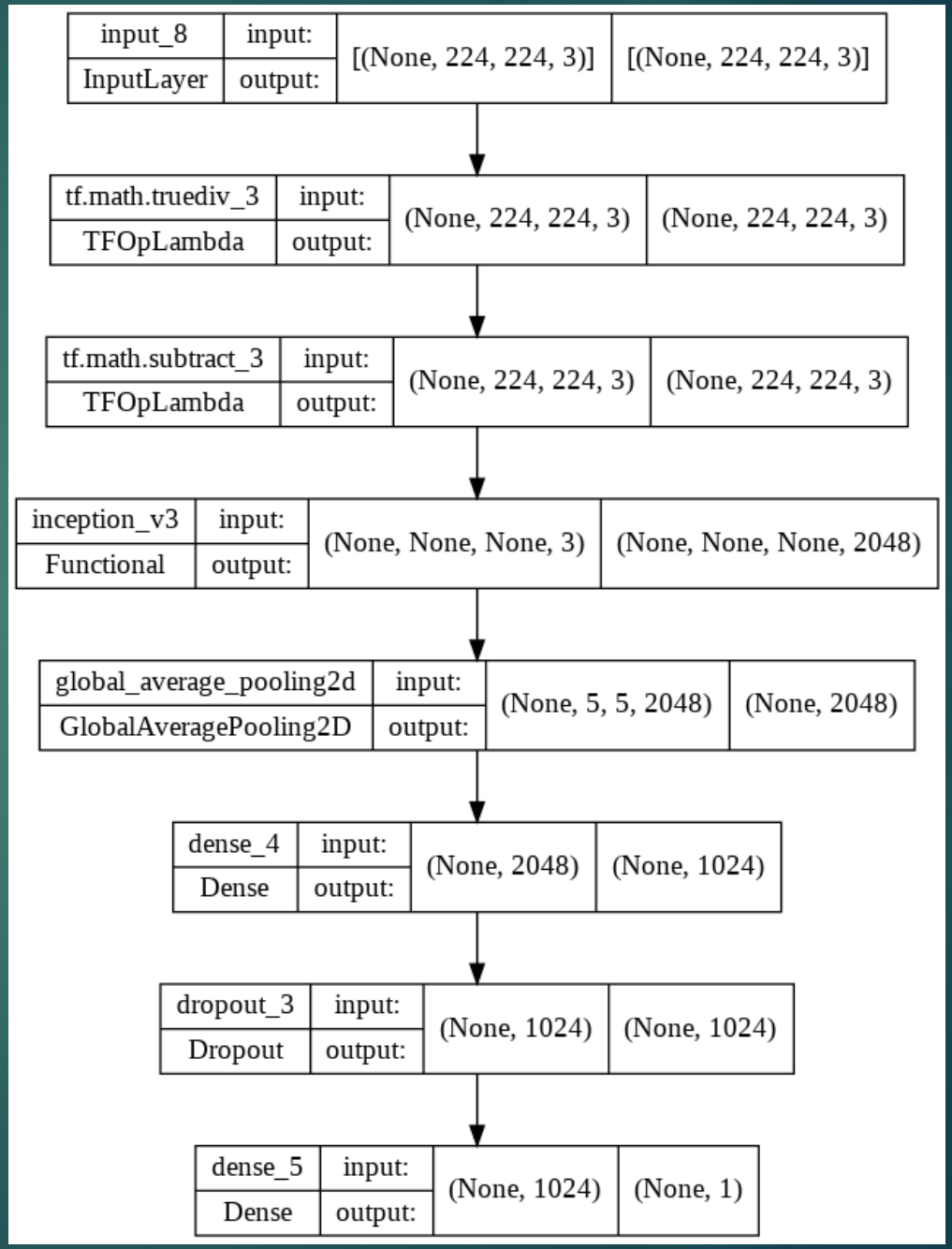
Inception vital Preprocessing on Inputs

- Each Keras Application expects a specific kind of input preprocessing. For InceptionV3, call tf.keras.applications.inception_v3.preprocess_input on your inputs before passing them to the model.
  - inception_v3.preprocess_input will scale input pixels between -1 and 1.

Output (prediction Layer)

- Since we have 2 classes, we needs a dense layer with one output.
- We chose “sigmoid” as Activation Function. This gives up the probability of each class. For showing output we need to have a threshold to determine which class the output belongs to.

Our Model Structure



DATA

Data

- consists of 25,077 images
  - ✓ organic (13,966)
  - ✓ recyclable (11,111)
    - Test data - 2513 images
    - Training data – 22500 images
- acquired images are coloured
- Resolution ranging from 191 pixels (minimum) to 264 pixels (maximum)
- total of 24,705 images have RGB colour mode

  uses 3 channels to represent Red-Green-Blue
- 372 images have P mode

  uses one channel

Downsizing the data

- We resized images to 224 × 224 pixels
- downsizing the original images for the following reason:
  1. a light-weight application for low-cost device with limited memory capacity and low-resolution camera

Gathering the data

- was based on **Sekar's** waste classification dataset available on Kaggle
- randomly portrait and landscape orientation

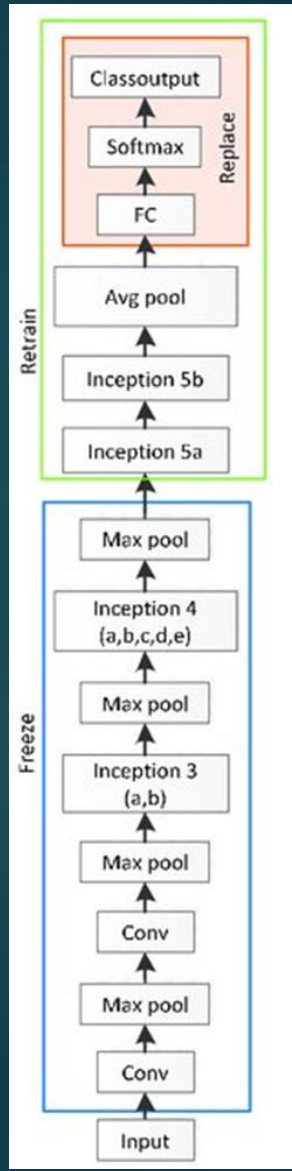
Training Phase

Training Phase different kinds

- Generally we can divide training phase in to two different kinds.
  - *“with pretrained weights”* or *“Transfer Learning”*
  - *“without pretrained weights”* or *“Fine Tuning”.*
- In first section, we goes with fine tuning and let all part of the model to be trained, so we left “weighted” option.
- In second one, we use ImageNet weights. In this case we need to freeze some layers (convolution layers) because we don’t want to change their wights. Then we train other parts. At last, We train top layers.

Transfer Learning Using GoogLeNet

- create the base pre-trained model
  - ✓ add a global spatial average pooling layer
  - ✓ add a fully-connected layer and a logistic layer
- this is the model we will train model
  - ✓ train only the top layers (freeze all convolutional InceptionV3 layers)
- train the model on the new data for a few epochs
  - ✓ top layers are well trained
  - ✓ fine-tuning convolutional layers from inception V3
  - ✓ freeze the bottom N layers and train the remaining top layers
  - ✓ train the top 2 inception blocks(freeze the first 249 layers and unfreeze the rest)
- train our model again
  - ✓ fine-tuning the top 2 inception blocks alongside the top Dense layers



Fine-tuning GoogLeNet

- Load Data
- Load Pretrained Network
- Replace Final Layers
- Train Network
- Classify Validation Images

Metrics

AUC & ACCURACY

ROC curve

- An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:
  - True Positive Rate
  - False Positive Rate
- True Positive Rate (TPR) is a synonym for recall and is therefore defined as follows:

$$TPR = \frac{TP}{TP + FN}$$

- False Positive Rate (FPR) is defined as follows:

$$FPR = \frac{FP}{FP + TN}$$

Continue…

An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows a typical ROC curve.

To compute the points in an ROC curve, we could evaluate a logistic regression model many times with different classification thresholds, but this would be inefficient. Fortunately, there's an efficient, sorting-based algorithm that can provide this information for us, called AUC.

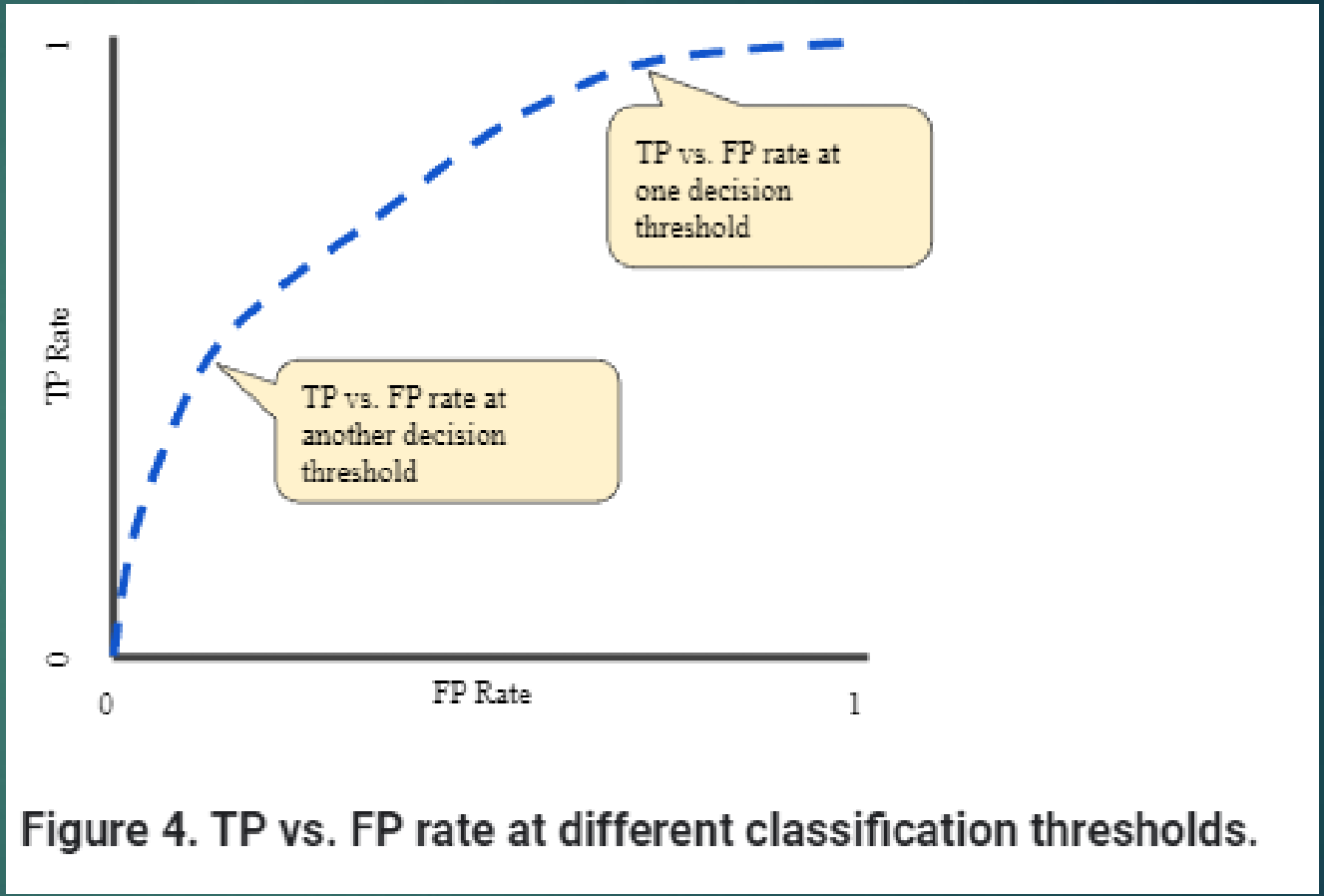


Figure 4. TP vs. FP rate at different classification thresholds.

AUC: Area Under the ROC Curve

AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1).

AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example. For example, given the following examples, which are arranged from left to right in ascending order of logistic regression predictions:

AUC represents the probability that a random positive (green) example is positioned to the right of a random negative (red) example.

AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.

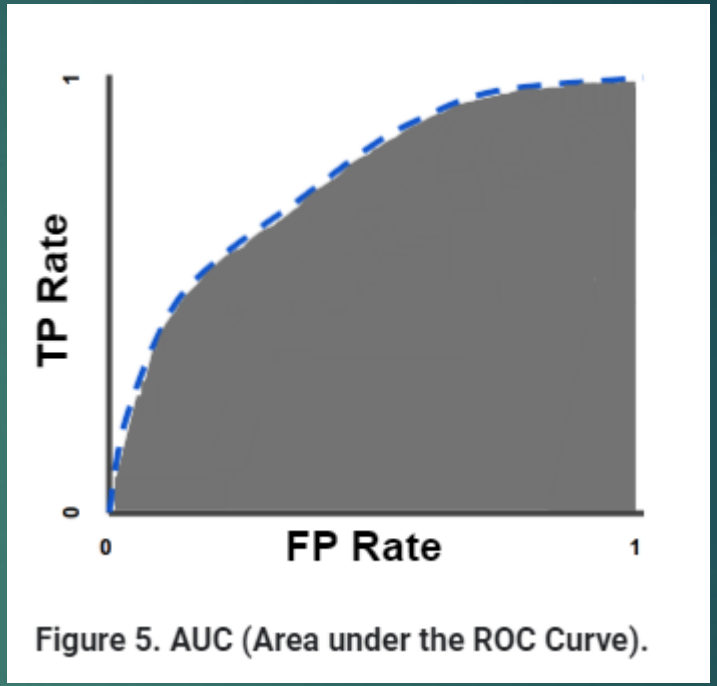
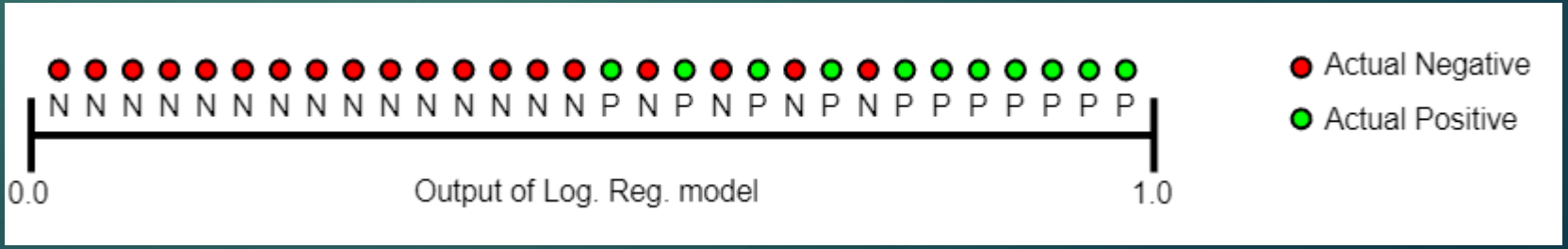


Figure 5. AUC (Area under the ROC Curve).



Accuracy

- Accuracy is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition:

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

- For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- Where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives.

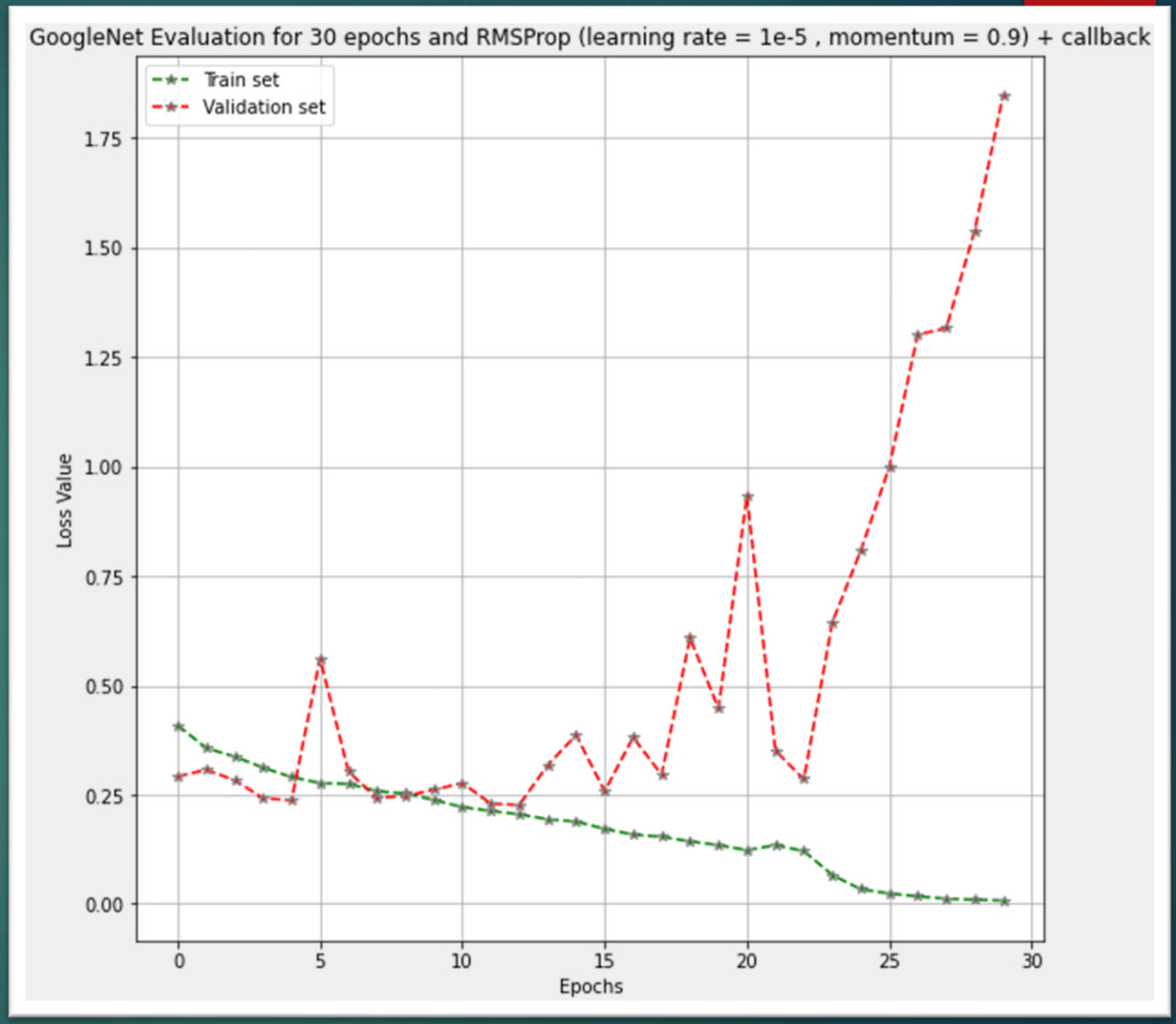
Evaluation

GoogleNet

Without early stopping

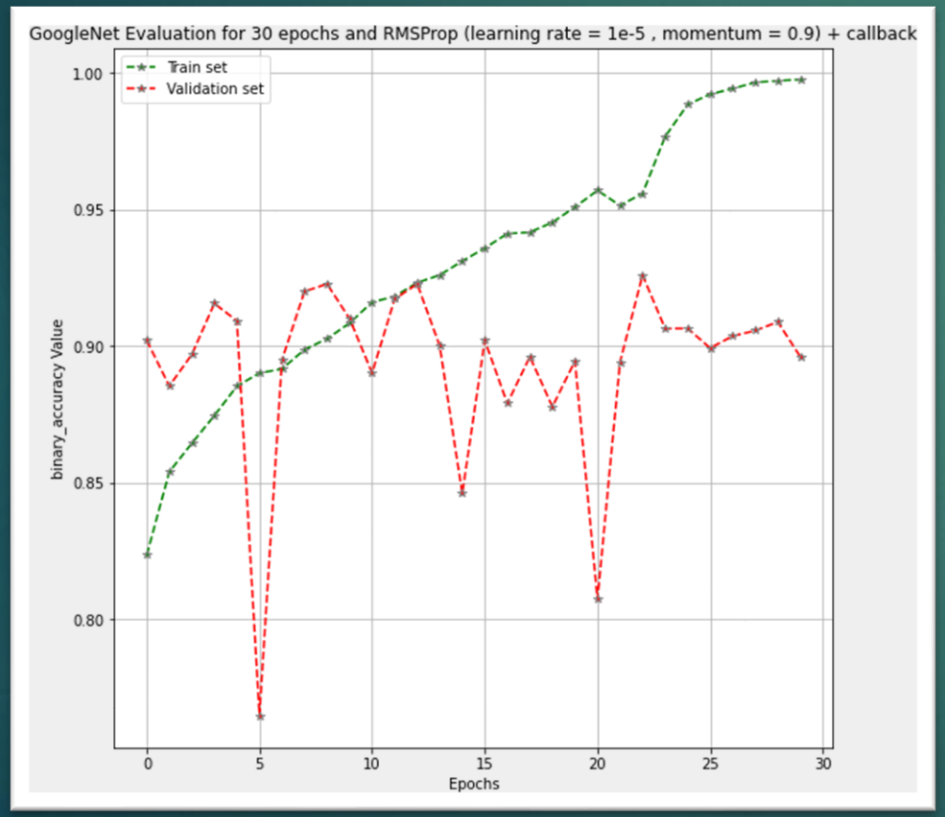
Loss plot

Model could do well although by increase in epochs, model started to get much more worse than before. We have to use some methods to stop model getting worse.

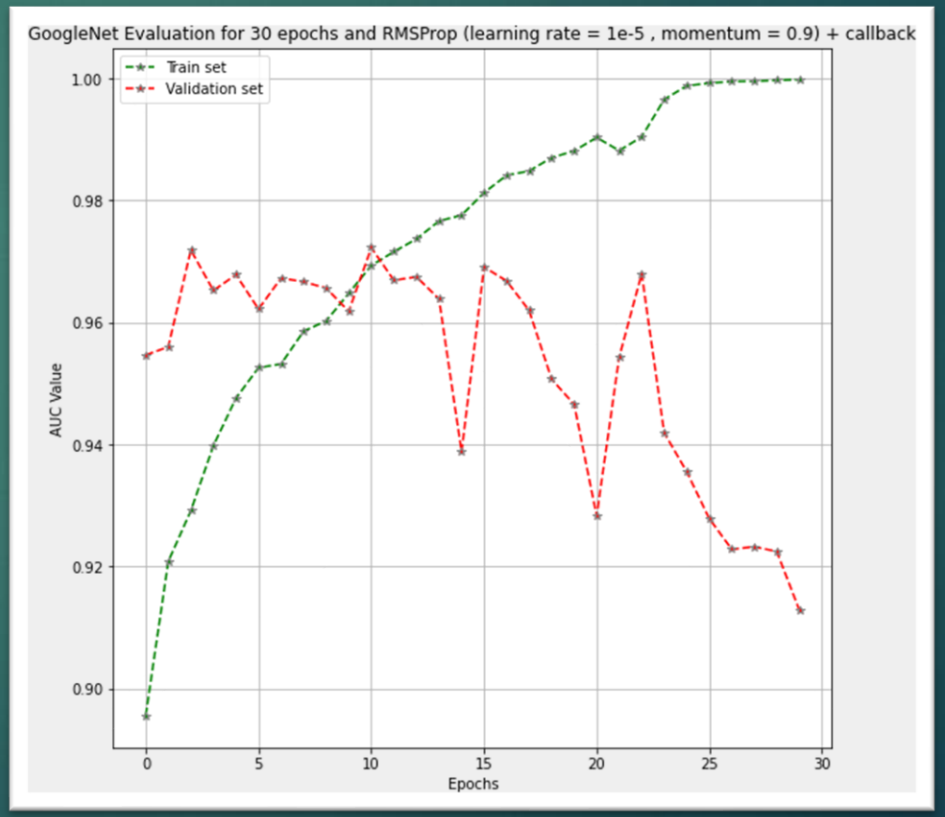


Metrics plot

Binary accuracy



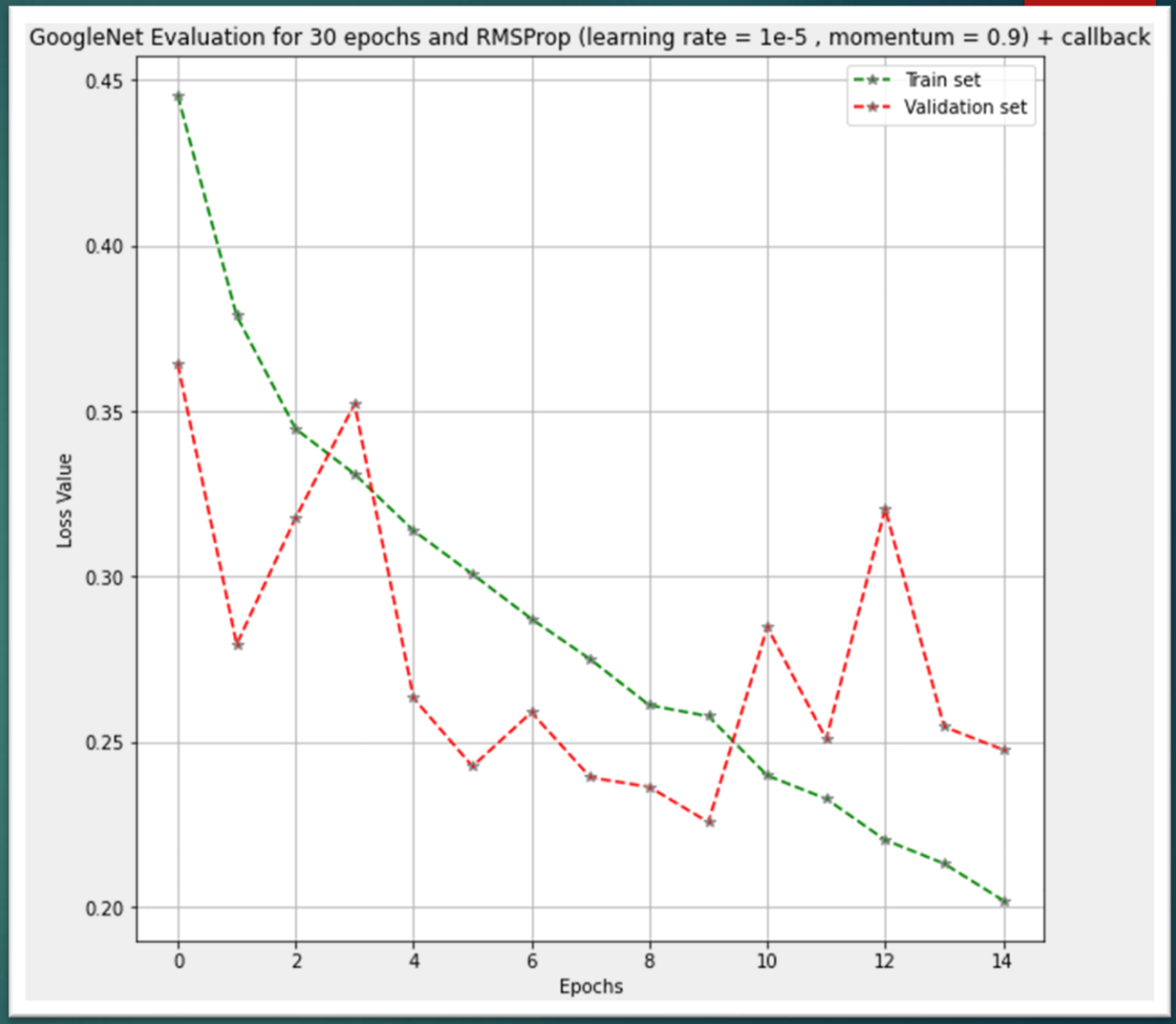
AUC



With early stopping

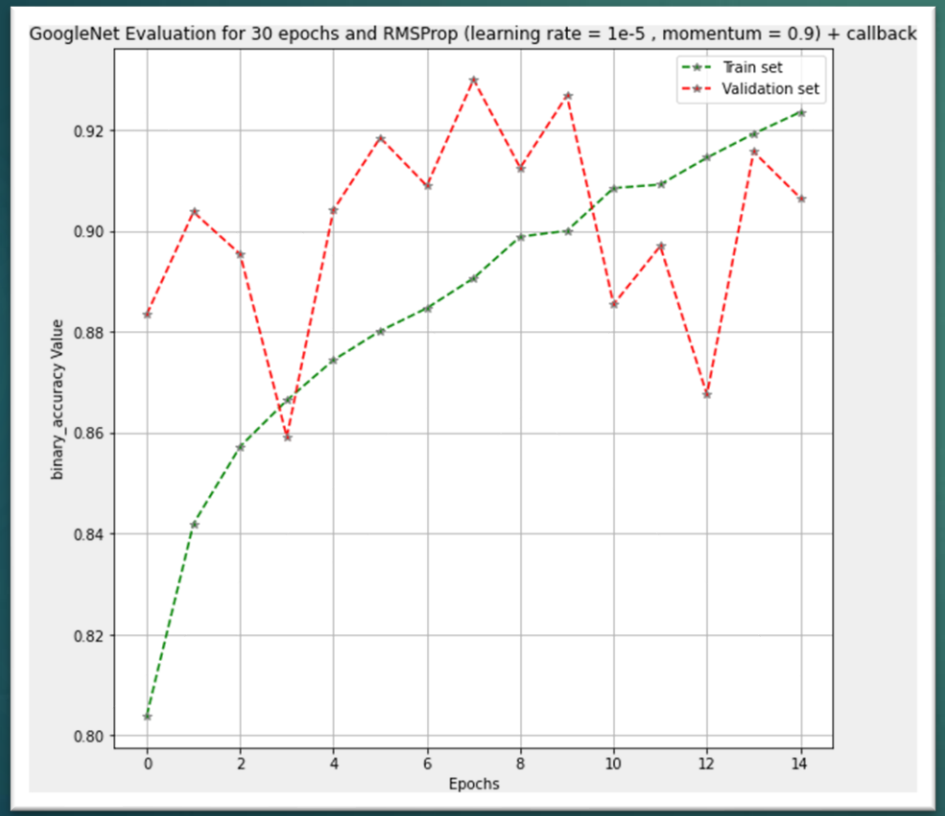
Loss plot

Early stopping stopped model at 14 epochs due to overfitting preventive.

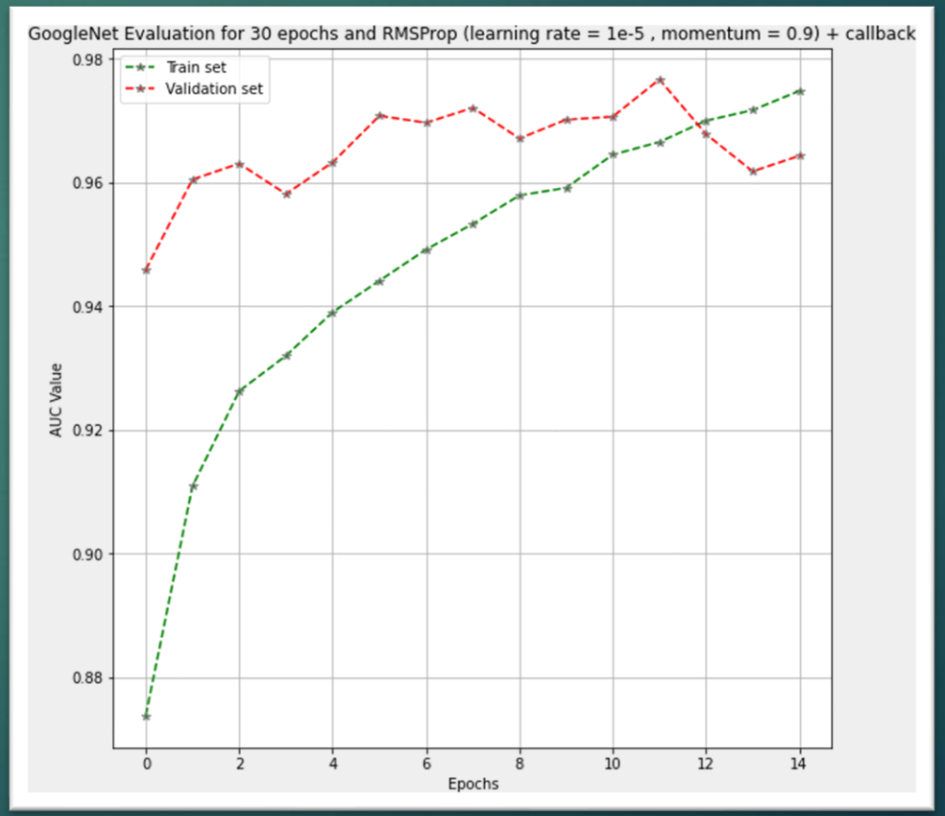


Metrics Plot

Binary accuracy



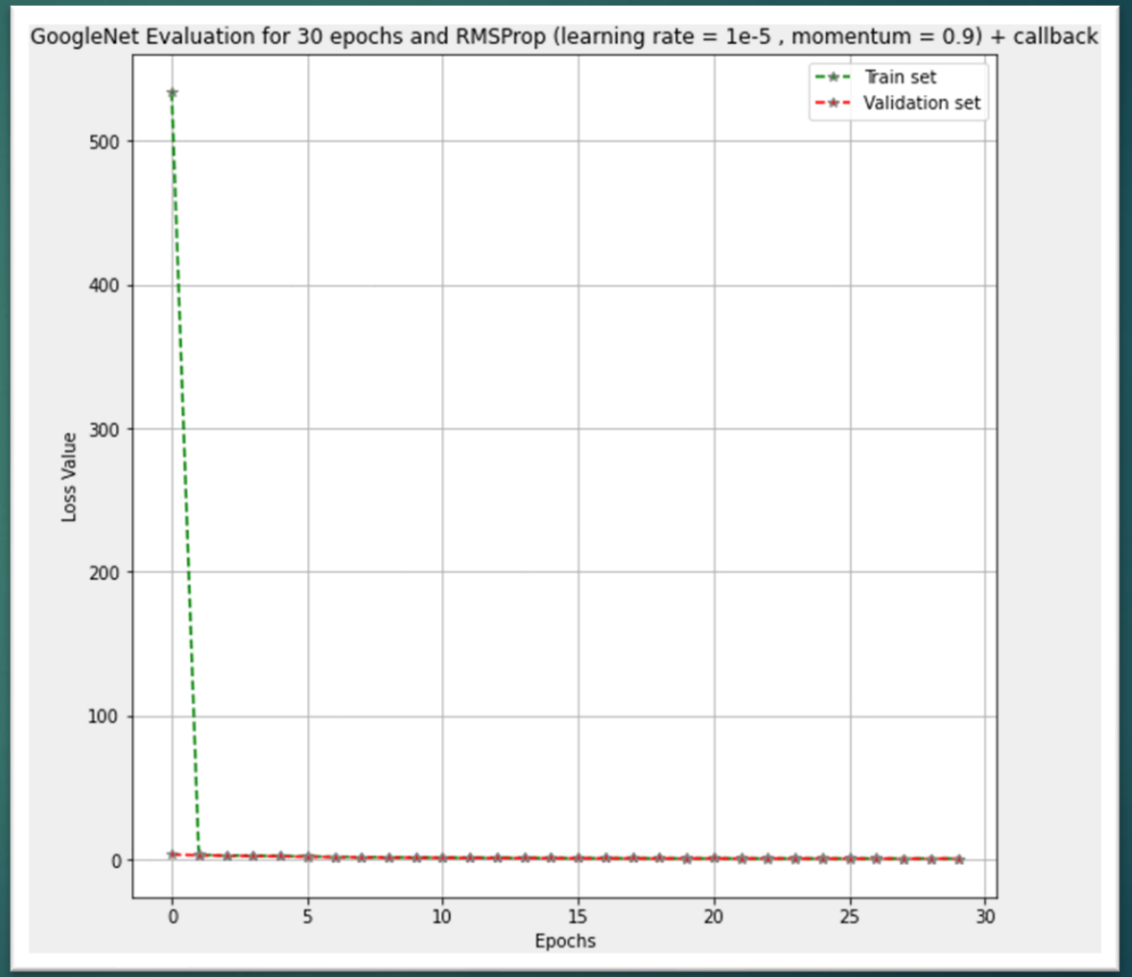
AUC



With regularization

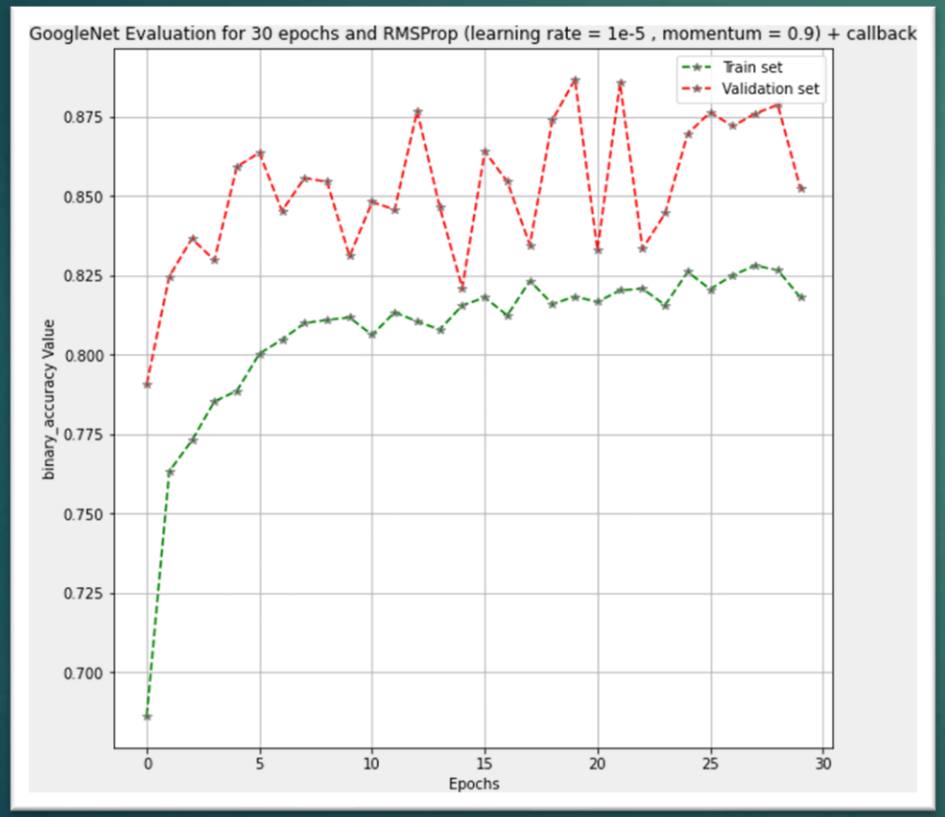
Loss plot

Unfortunately, regularization term couldn't do well. There are some options to improve model performance. We can use test different other type of regularization or we can omit some of them.

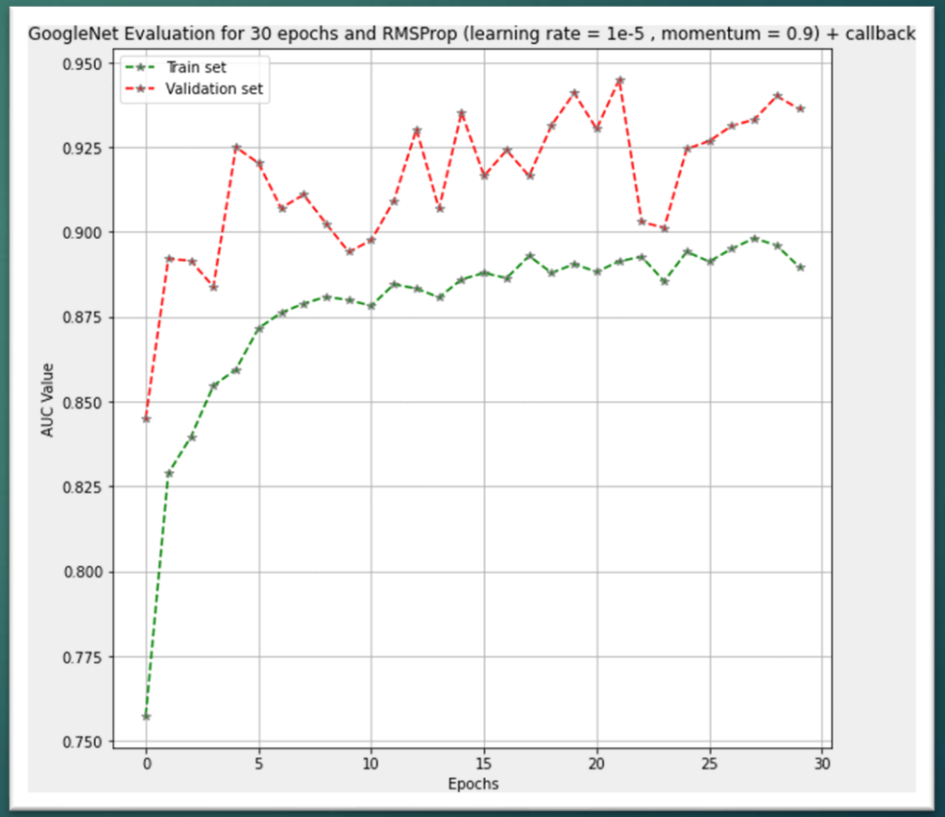


Metrics Plot

Binary accuracy



AUC



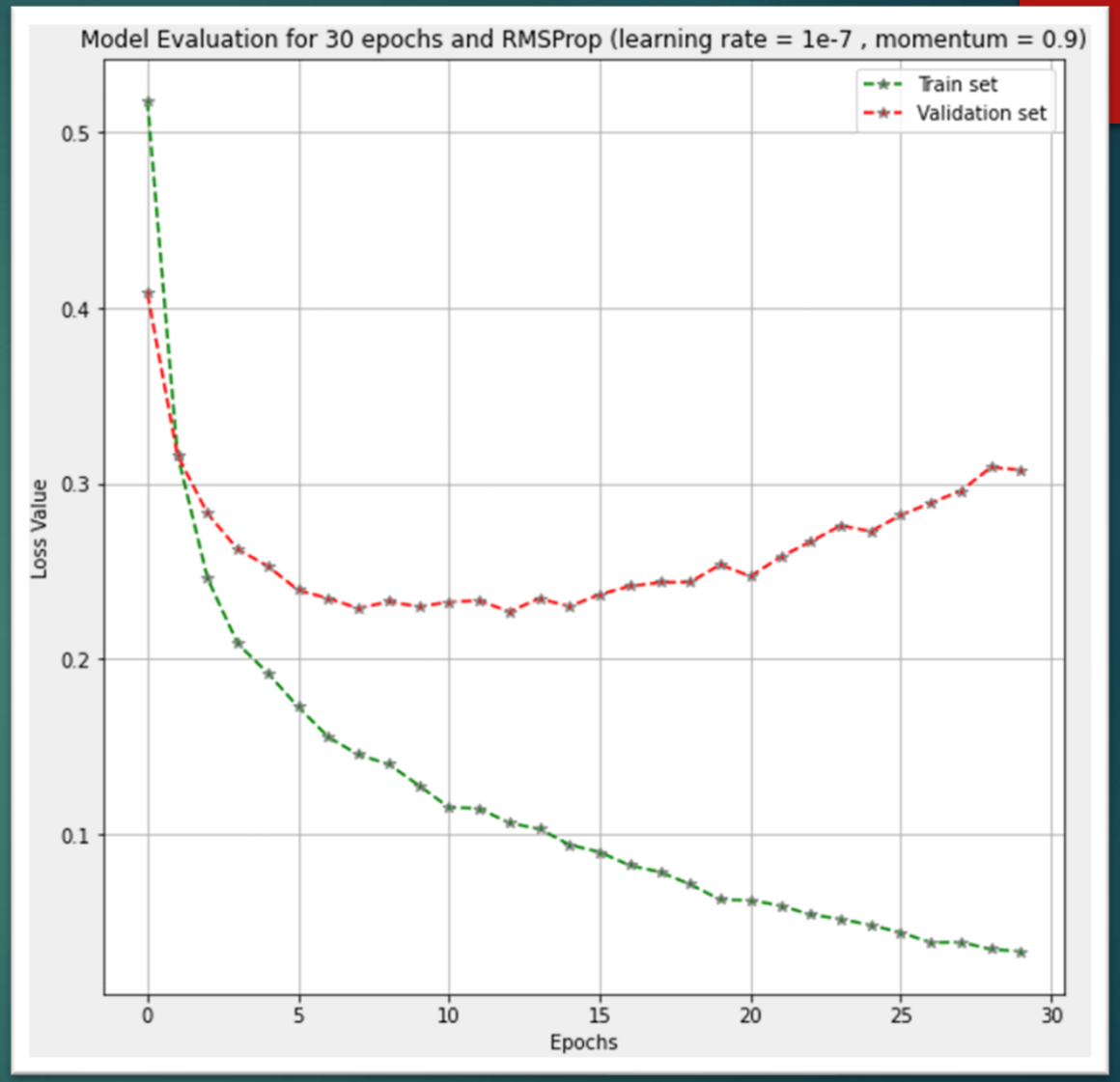
Inception V3

Fine tuning

WITHOUT EARLY STOPPING

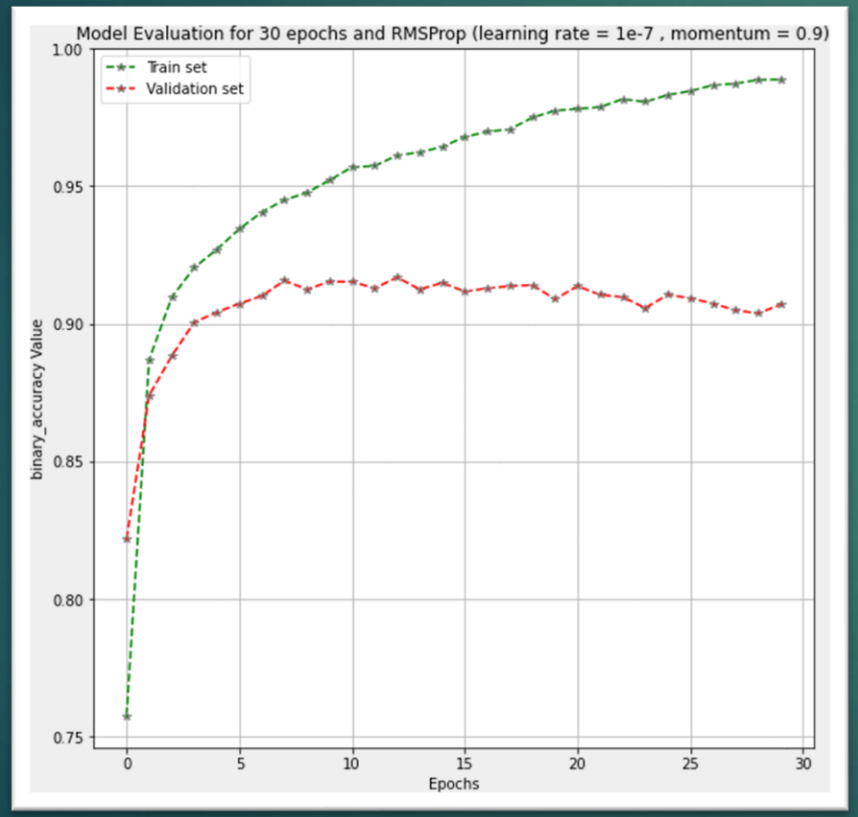
Loss Plot

As it can be seen, the model has started to overfit. So it needs to stop. For this, we can set an early stop point.

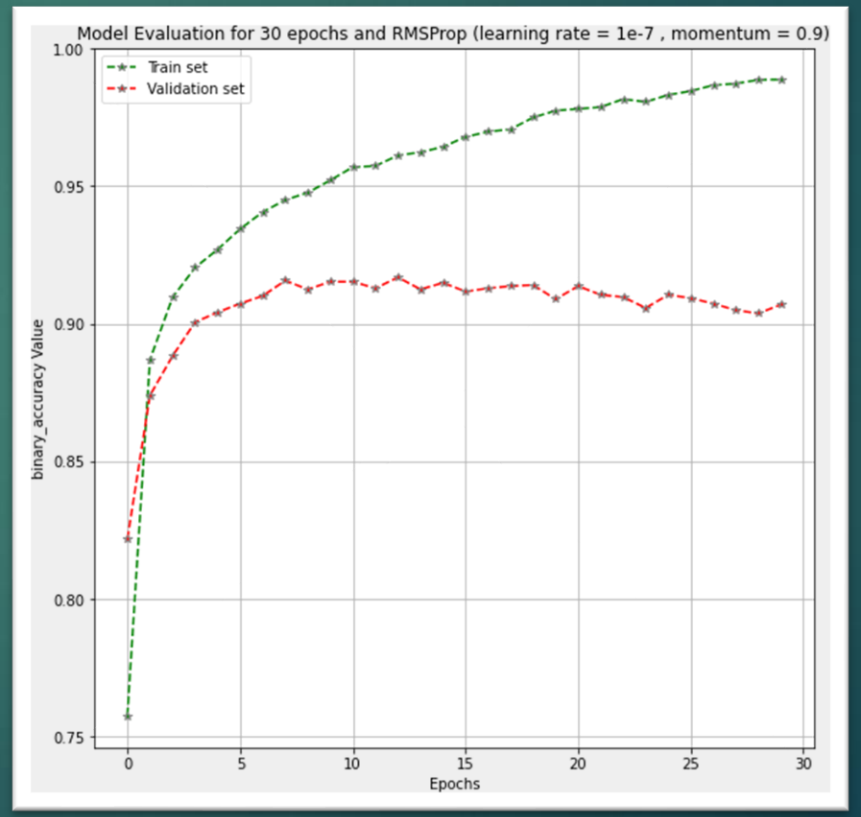


Metrics Plot

Binary accuracy



AUC

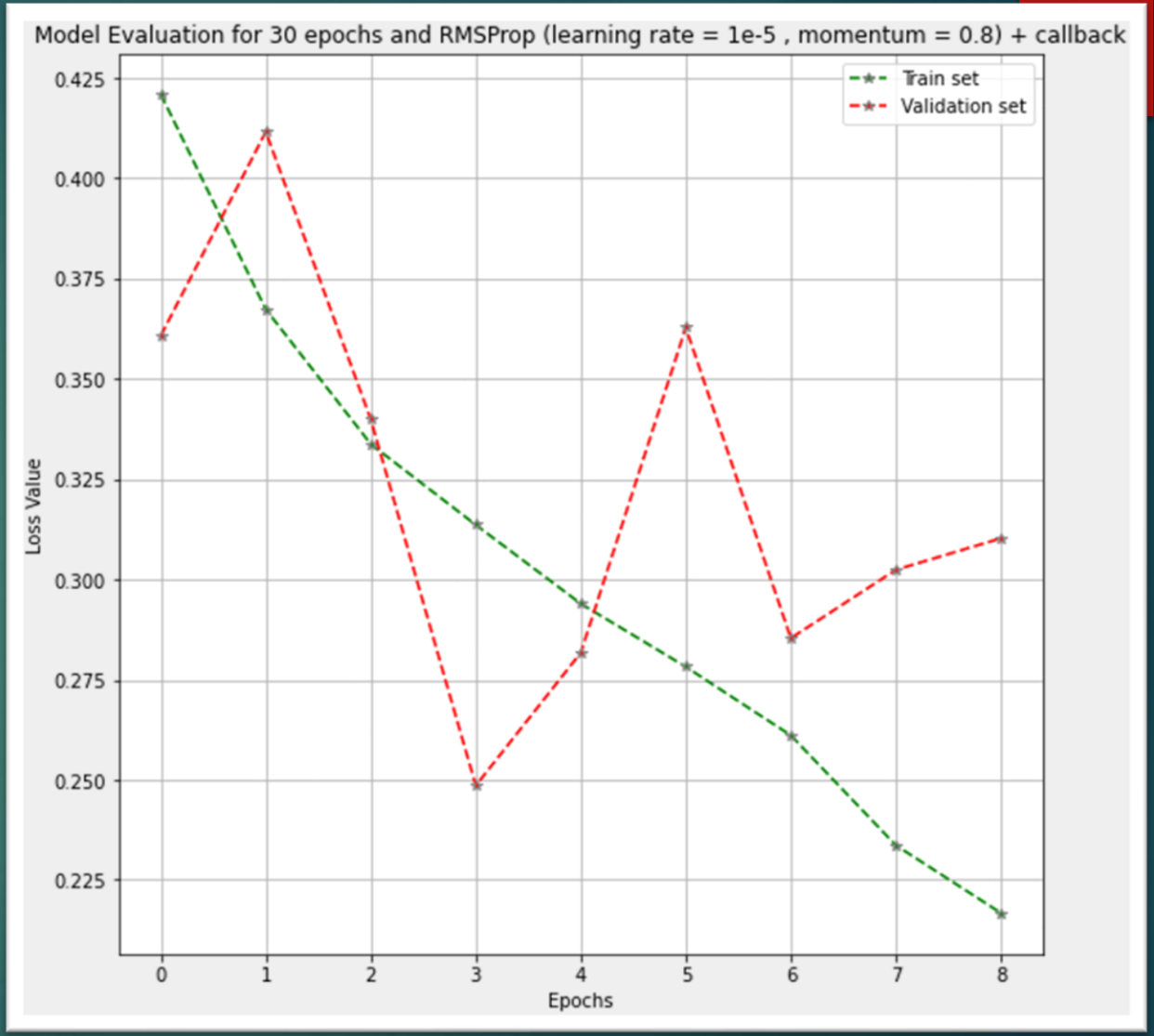


Fine tuning

WITH EARLY STOPPING

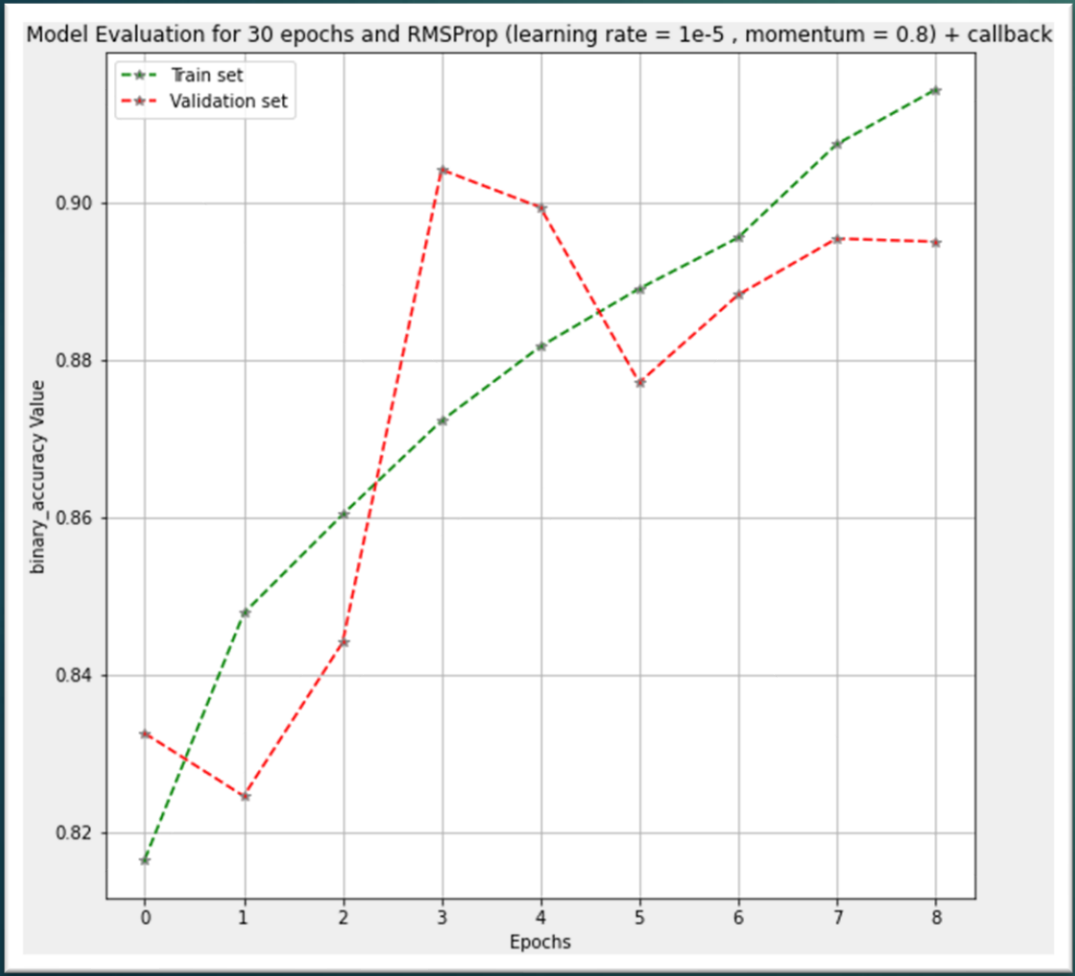
Loss Plot

Strategies makes the model stop at end of first epoch. It means if we go further, there is likely to be an overfitting.

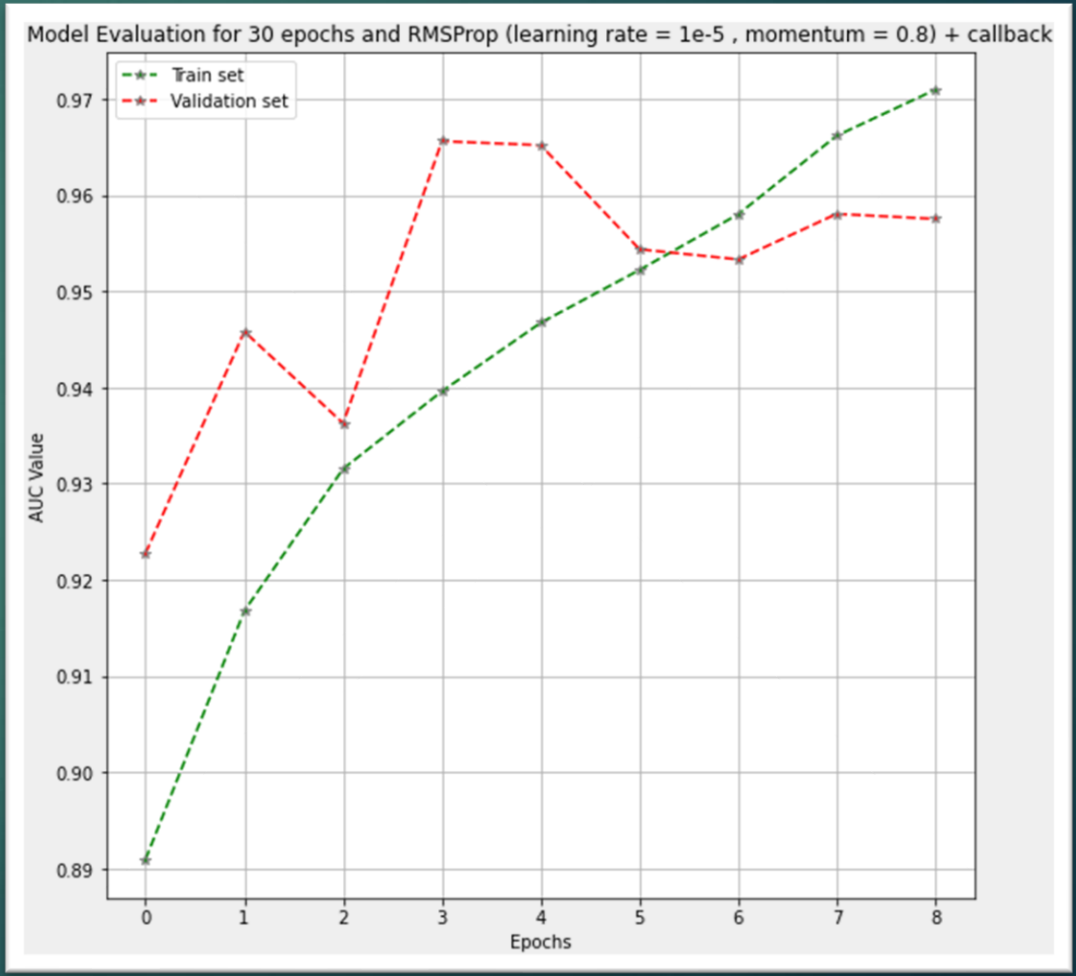


Metrics Plot

Binary Accuracy



AUC

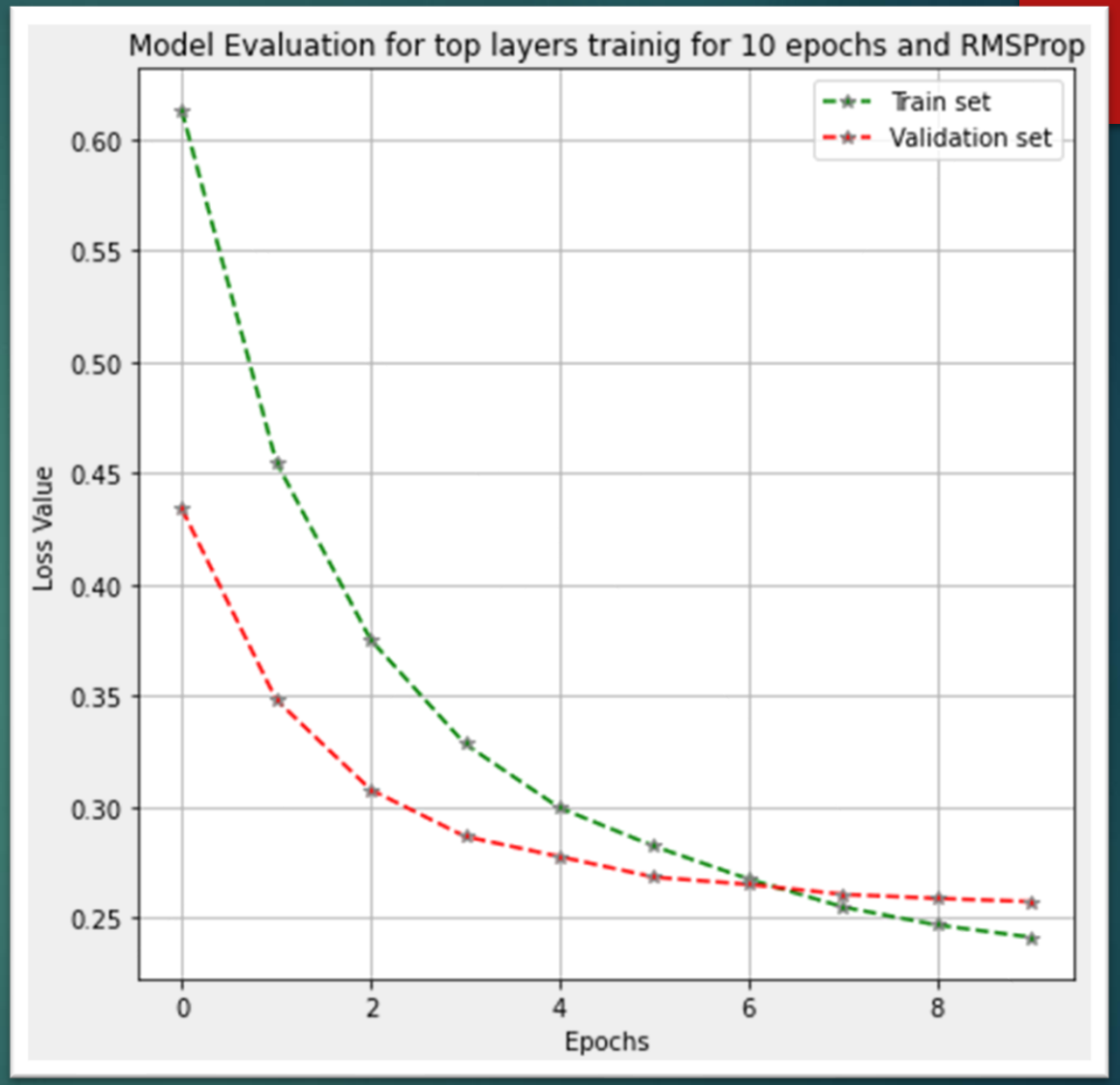


Transfer Learning

Top layers

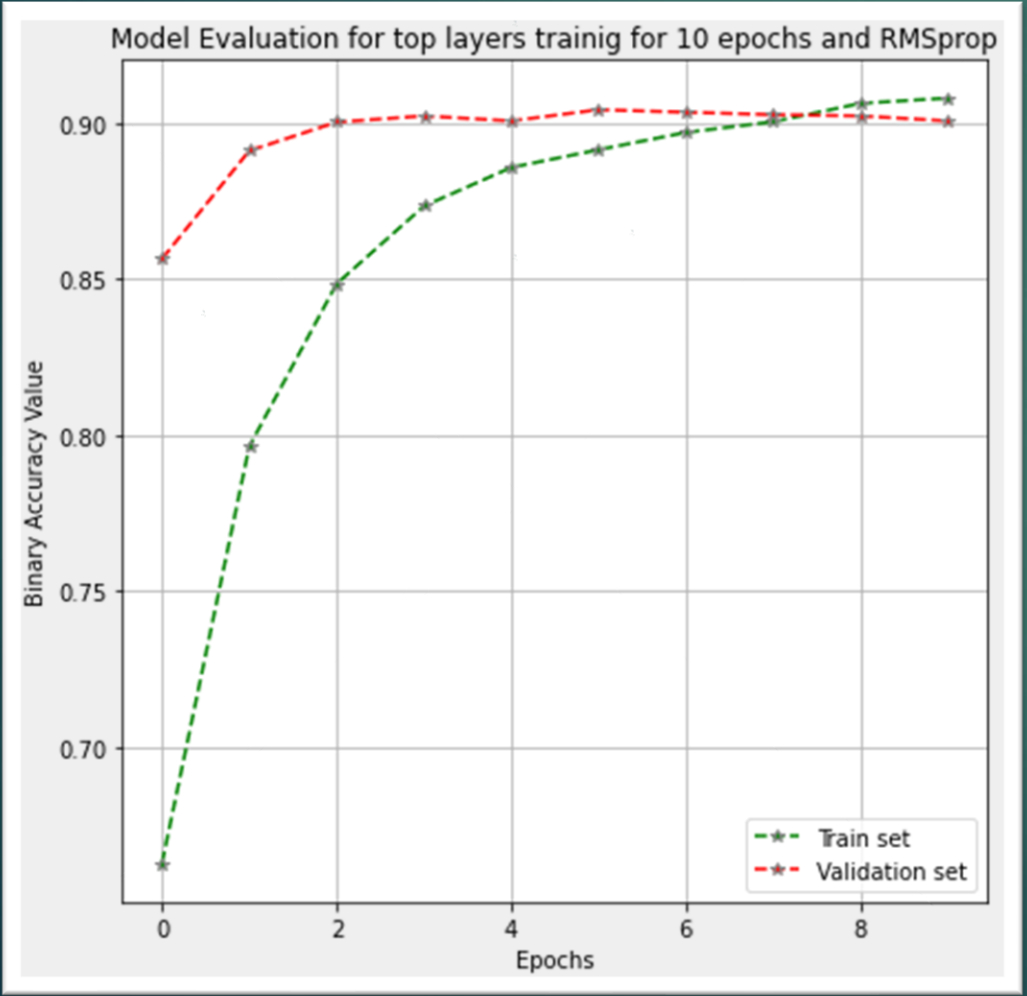
Loss Plot

Model overfitted from epoch 5 so it should be stopped on this epochs. Adding callback would help the model.

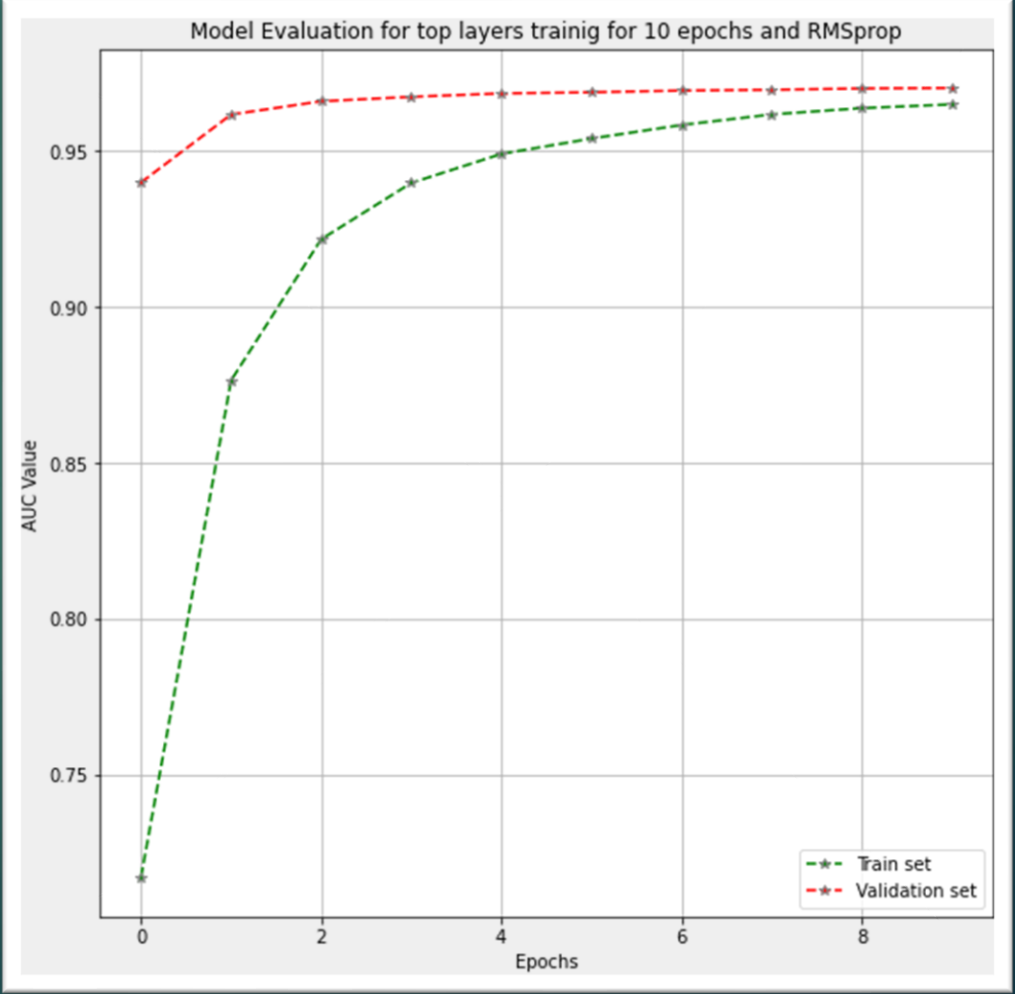


Metrics Plot

Binary accuracy



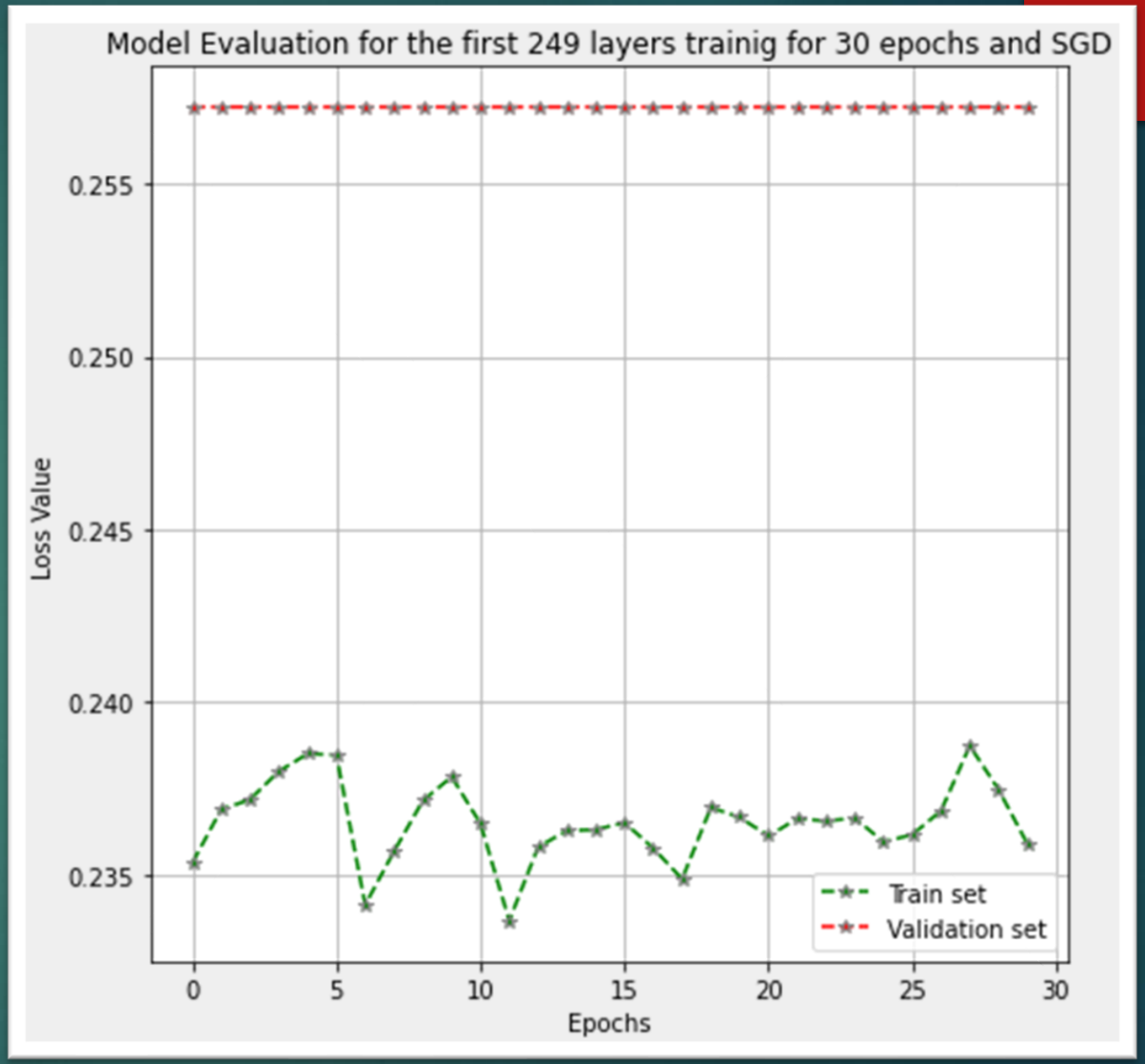
AUC



the top 2 inception blocks

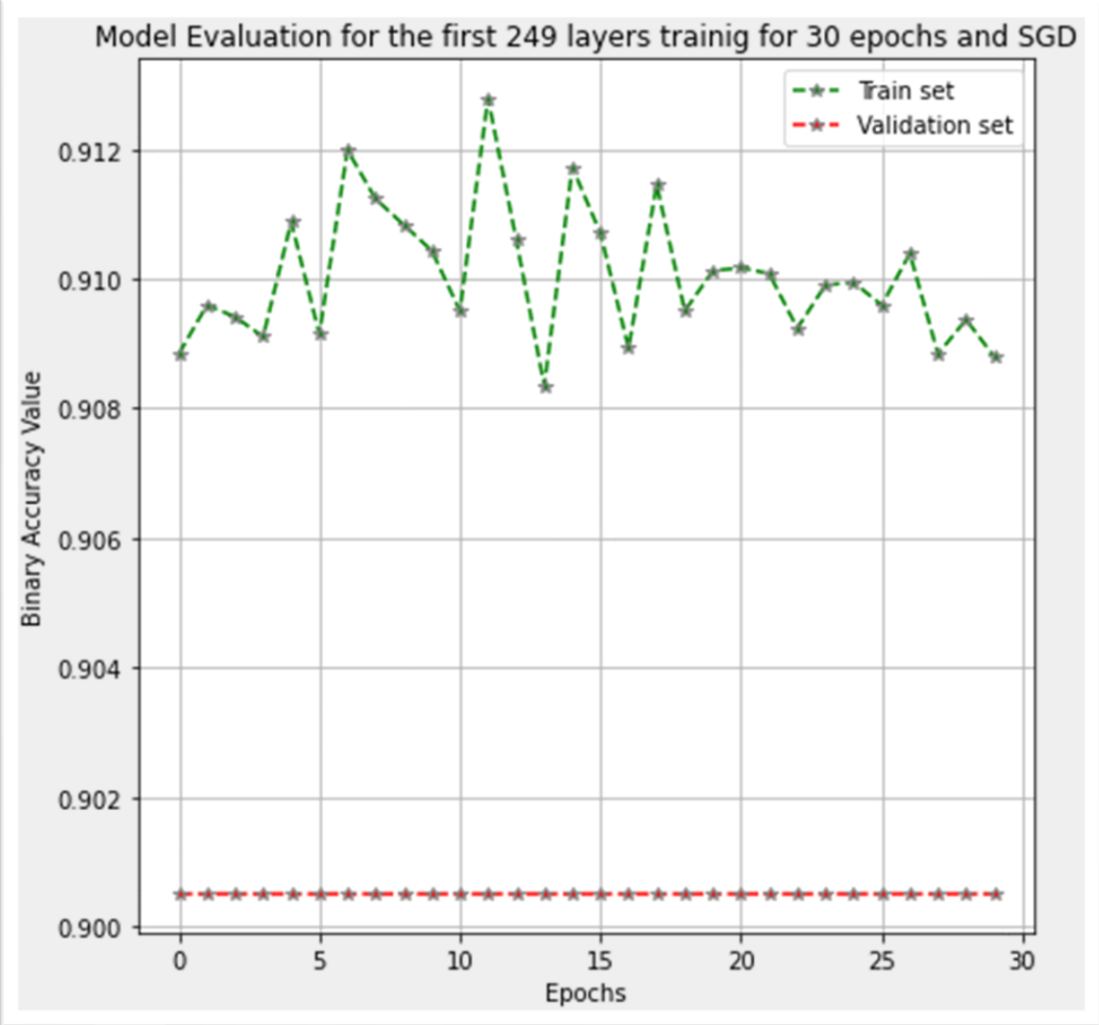
Loss Plot

Here we used RMSprop (title is not correct). Model couldn't perform correctly. This means changing in the training phase is needed. For alternative, we can choose other layers/blocks to be learnt.

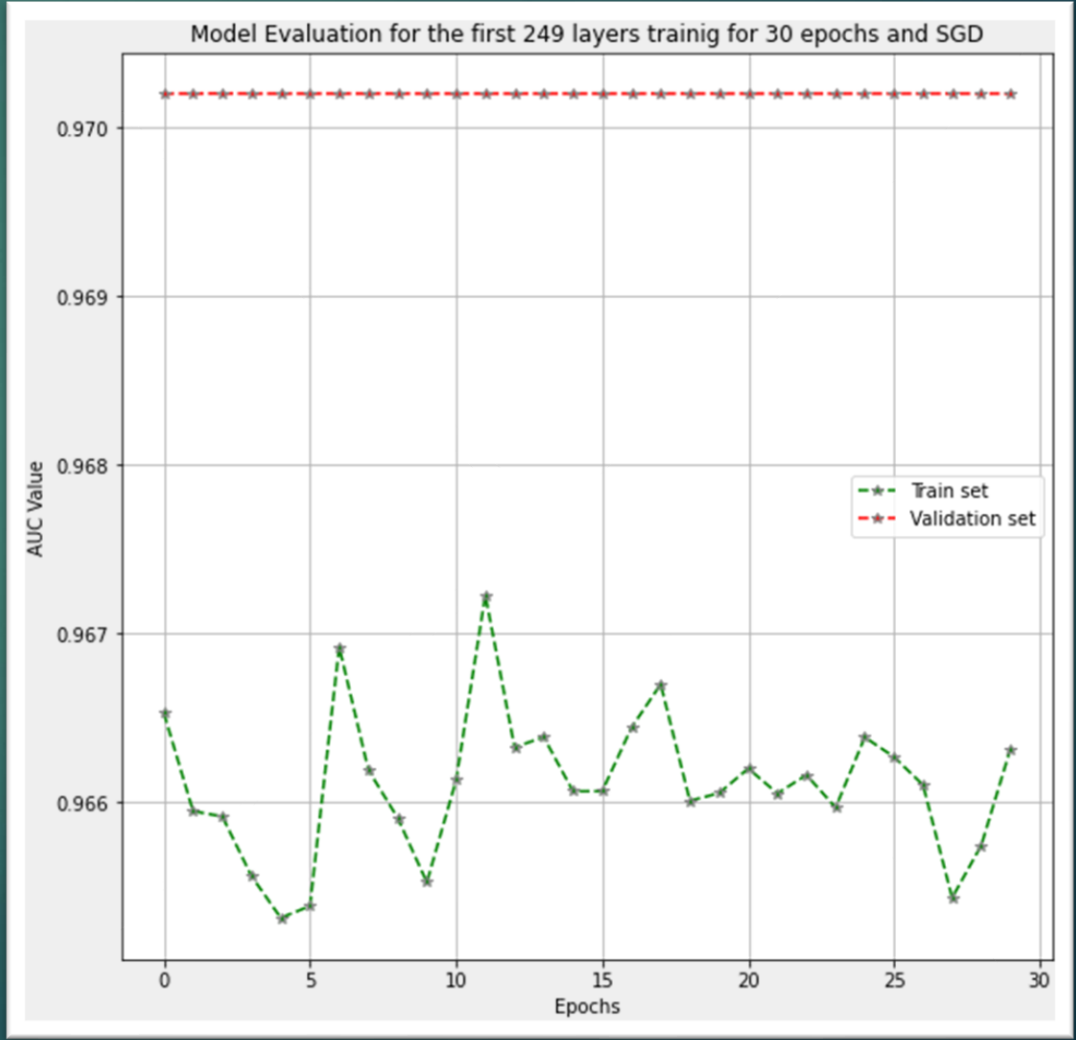


Metrics Plot

Binary accuracy



AUC



Result

- When it comes to GoogleNet (first version), when it was trained without early stopping, it got overfitted.
- Using early stopping prevented overfitting and caused model reaching around 90% in accuracy and around 0/95 in AUC. As it can be seen, adding regularization term couldn't improve model performance.
- Inception v3 which is the newer version of GoogleNet, could do much more better than it's first generation.
- In transfer learning, It could reach just over 90% as accuracy and over 0/95 as AUC. Model could reach higher with pretrained layers but need changes in training strategies.
- In Fine Tunning model had the best performance , although it stock in overfitting after a few epochs. Using some methods which was mentioned before made significant changes in models performance.
- Note: All recorded values are for validation set.

References

1. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., … Rabinovich, A. (2015). Going deeper with convolutions. Proceedings of the IEEE conference on computer vision and pattern recognition
2. https://d2l.ai/chapter_convolutional-modern/googlenet.html
3. Sekar, S. Waste Classification Data, Version 1. Available online: https://www.kaggle.com/techsash/waste-classification-data
4. Nnamoko, N.; Barrowclough, J.; Procter, J. Solid Waste Image Classification Using Deep Convolutional Neural Network. Infrastructures 2022, 7, 47.
5. Malik, M.; Sharma, S.; Uddin, M.; Chen, C.-L.;Wu, C.-M.; Soni, P.; Chaudhary, S. Waste Classification for Sustainable Development Using Image Recognition with Deep Learning Neural Network Models. Sustainability 2022, 14, 7222.
6. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 2818-2826, doi: 10.1109/CVPR.2016.308