International Journal of Computer Science Trends and Technology (IJCST) – Volume 11 Issue 2, Mar-Apr 2023

RESEARCH ARTICLE OPEN ACCESS

Hyperparameter Tuning for Convolution Neural Network

Dr. Girish Tere [1], Mr. Kuldeep Kandwal [2]

[1] Department of Computer Science, Thakur College of Science & Commerce, Mumbai, Maharashtra, India
[2] Department of Mathematics, Thakur College of Science & Commerce, Mumbai, Maharashtra, India

ABSTRACT

Limitation of Artificial Neural Network can be overcome by using Convolutional Neural Network (CNN). CNN will work in better way if images/videos need to be processed. This research work attempts to develop CNN model for image classifier using different hyperparameters. Problem statement is how o tune these hyperparameters so that developed model will show less loss or better accuracy. There are many libraries available for hypertuning such as KearsTuner. It is observed that customizing tuning hyperparameters manually is better approach as each library has its own limitations.

Keywords: - CNN, hyperparameters, convolution layers, filters, convolution layer, pooling, flatten layer, padding, stride

I. INTRODUCTION

Convolutional neural networks leverage spatial information, and they are therefore very well-suited for classifying images [1, 4].

A. Preserves Spatial orientation
B. Reduces learnable parameters

II. METHODOLOGY

CNN model is built to solve Emergency vs non-Emergency vehicle classification problem[2]. Following steps are completed for this experiment. [1]

1. Loading the Dataset
2. Pre-processing the Data
3. Creating Training and Validation set
4. Defining the Model Architecture
5. Compiling the Model
6. Training the Model
7. Evaluating model performance

III. EXPERIMENTAL SETUP

CNN Model is developed for classification of Emergency and non-emergency vehicles. The notebook is executed in Colab using GPU. Various libraries such as TensorFlow, Keras are imported.

A. *Construction of CNN*

1. Loading the Dataset

Following libraries and functions were used:

```
numpy as np
pandas as pd
matplotlib.pyplot as plt
%matplotlib inline
from keras.layers import Dense, InputLayer,
BatchNormalization, Dropout
from keras.models import Sequential
from keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

2. Pre-processing the Data

Since CNN is used, images resizing into 1D array is not required. Normalizing the pixel values is performed.

3. Creating Training and Validation set

Training and validation sets are created using train_test_split method. Training dataset was 70% and Testing dataset was 30% used. Shape of training and validation set are

(((1646, 224, 224, 3), (1646,)), ((706, 224, 224, 3), (706,)))

4. Defining the Model Architecture

Imported the convolutional and flatten layer from Keras. Following Model architecture is developed for the experiment.

1. Used Keras sequential model

2. Defined input layer with 3D input of shape (224,224,3)

3. Defined the first convolutional layer with 25 filters of size (5,5), used Relu activation function, strides (1,1) and 'valid' padding is used.

4. Defined the second convolutional layer with 50 filters of size (5,5), used Relu activation function, strides (1,1) and ‘valid’ padding is used.

5. Flattened the output from convolutional layers so that it can be forwarded to the dense layers

6. Defined the first dense or fully connected layer with 100 neurons

7. Defined the output layer with 1 neuron since it is a binary classification problem

Summary of the model1:

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 220, 220, 25)	1900
conv2d_2 (Conv2D)	(None, 216, 216, 50)	31300
flatten_1 (Flatten)	(None, 2332800)	0
dense_1 (Dense)	(None, 100)	233280100
dense_2 (Dense)	(None, 1)	101

Total params: 233,313,401
Trainable params: 233,313,401
Non-trainable params: 0

5. Compiling the Model

Compiled the model using

model.compile(loss='binary_crossentropy', optimizer="sgd", metrics=['accuracy'])

6. Training the Model

Model is trained using following command:

model_history = model.fit(X_train, y_train, epochs=10, batch_size=128,validation_data=(X_valid,y_valid))

7. Evaluating model performance

Accuracy on training set: 0.7612393681652491 %

Accuracy on validation set:0.7195467422096318 %

Fig. 1 show summarize history for loss

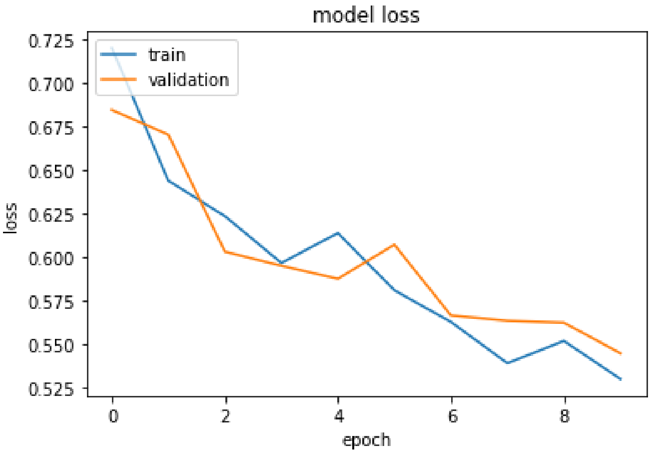

Fig. 1: Model Loss

Fig. 2 show summarize history for accuracy

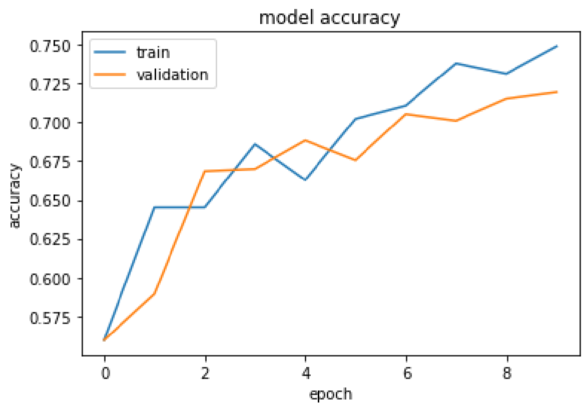

Fig. 2: Model Accuracy

B. *Adding Maxpool layer to CNN to rereduce parameters*

To achieve this maxpool layer from keras is imported. New CNN model is developed, adding maxpool layer after the convolutional layers. Summary of the model2:

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 220, 220, 25)	1900
conv2d_4 (Conv2D)	(None, 216, 216, 50)	31300
max_pooling2d_1 (MaxPooling2	(None, 54, 54, 50)	0
flatten_2 (Flatten)	(None, 145800)	0
dense_3 (Dense)	(None, 100)	14580100
dense_4 (Dense)	(None, 1)	101

Total params: 14,613,401
Trainable params: 14,613,401
Non-trainable params: 0

Training the Model

Model is trained using following command:

model_history = model.fit(X_train, y_train, epochs=10, batch_size=128,validation_data=(X_valid,y_valid))

C. *Hyperparameter Tuning of CNN*

Among the diverse deep learning architecture, convolutional neural network stands out for its unprecedented performance on computer vision. Tuning hyperparameters for deep neural network [5] is difficult as it is slow to train a deep neural network and there are numerous parameters to configure. In this section, effect of change of various hyperparameters for convnet are observed. Activation Function was changed to Sigmoid. Rest of Model architecture is same as explained in Model 1.

compiling the model with maxpool layer

model.compile(loss='binary_crossentropy', optimizer="sgd", metrics=['accuracy'])

Following hyperparameters are changed and for every change the CNN Model loss and Model accuracy is measured.

1. increase number of convolutional layers
2. increase number of pooling layers
3. increase number of convolutional filters
4. change size of convolutional filters
5. change pooling type
6. change padding technique
7. change stride

Hyperparameters of fully connected layers
1. change activation function of hidden layer
2. increase hidden neurons

3. increase hidden layers
4. increase number of epochs
5. change optimizer
6. add batch normalization layer
7. add dropout layer

Different CNN models were developed for same emergency vehicle dataset with different hyperparameters and in next session results obtained are discussed.

IV. RESULTS AND DISCUSSION

Following Keras sequential models are developed and they are evaluated.

Model	Input Layer	First Convolution Layer	second convolutional layer	Pooling Layer	Flattened the output from convolutional layers?	first dense or fully connected layer	Output Layer with activation function sigmoid	Loss and Optimizer	No. Of Epochs	Batch Size
Model1	3D input of shape (224,224,3)	25 filters of size (5,5), used Relu activation function, strides (1,1) and 'valid' padding	50 filters of size (5,5), used Relu activation function, strides (1,1) and 'valid' padding	Nil	Yes	100 neurons	1 neuron	loss='binary_crossentropy', optimizer="sgd",	10	128
Accuracy on training set: 0.7612393681652491 %, Accuracy on validation set: 0.7195467422096318 %										
Model2	3D input of shape (224,224,3)	25 filters of size (5,5), used Relu activation function, strides (1,1) and 'valid' padding	50 filters of size (5,5), used Relu activation function, strides (1,1) and 'valid' padding	Maxpool layer	Yes	100 neurons	1 neuron	loss='binary_crossentropy', optimizer="sgd",	10	128
Accuracy on training set: 0.6944106925880923 %, Accuracy on validation set: 0.6898016997167139 %										
Model3	3D input of shape (224,224,3)	25 filters of size (5,5), used Relu activation function, strides (1,1) and 'valid' padding	50 filters of size (5,5), used Relu activation function, strides (1,1) and 'valid' padding	Maxpool layer	Yes	100 neurons	First output layer: 100 neurons Second output layer: 1 neuron	loss='binary_crossentropy', optimizer="sgd",	10	128
Accuracy on training set: 0.6852976913730255 %, Accuracy on validation set: 0.6728045325779037 %										
Model4	3D input of shape (224,224,3)	25 filters of size (5,5), used Relu activation function, strides (1,1) and 'valid' padding + Maxpool layer	50 filters of size (5,5), used Relu activation function, strides (1,1) and 'valid' padding + Maxpool layer	2 Maxpool layers	Yes	100 neurons	1 neuron	loss='binary_crossentropy', optimizer="sgd",	10	128
Model5	3D input of shape (224,224,3) Increasing number of convolutional filters	50 filters of size (5,5), used Relu activation function, strides (1,1) and 'valid' padding	75 filters of size (5,5), used Relu activation function, strides (1,1) and 'valid' padding	Maxpool layer	Yes	100 neurons	1 neuron	loss='binary_crossentropy', optimizer="sgd",	10	128
Accuracy on training set: 0.7168894289185905 %, Accuracy on validation set: 0.6883852691218131 %										

Model6	3D input of shape (224,224,3) Changing size of convolutional filters	25 filters of size (3,3), used Relu activation function, strides (1,1) and 'valid' padding	50 filters of size (3,3), used Relu activation function, strides (1,1) and 'valid' padding	Maxpool layer	Yes	100 neurons	1 neuron	loss='binary_crossentropy', optimizer="sgd",	10	128
Accuracy on training set: 0.715674362089915 %, Accuracy on validation set: 0.7096317280453258 %										
Model6	**3D input of shape (224,224,3)** **Changing size of filters & pooling size**	**25 filters of size (3,3), used Relu activation function, pool_size=(2, 2) strides (1,1) and 'valid' padding**	**50 filters of size (3,3), used Relu activation function, pool_size=(2, 2) strides (1,1) and 'valid' padding**	**Maxpool layer**	**Yes**	**100 neurons**	**1 neuron**	**loss='binary_crossentropy', optimizer="sgd",**	**10**	**128**
Accuracy on training set: 0.735722964763062 %, Accuracy on validation set: 0.7209631728045326 %										
Model7	3D input of shape (224,224,3) Changing pooling size and Padding technique	25 filters of size (3,3), used Relu activation function, pool_size=(4, 4) strides (1,1) and 'same' padding	50 filters of size (3,3), used Relu activation function, pool_size=(4, 4) strides (1,1) and 'same' padding	Maxpool layer	Yes	100 neurons	1 neuron	loss='binary_crossentropy', optimizer="sgd",	10	128
Accuracy on training set: 0.7284325637910085 %, Accuracy on validation set: 0.7039660056657224 %										
Model8	3D input of shape (224,224,3) Changing stride	25 filters of size (3,3), used Relu activation function, pool_size=(4, 4) strides (2,2) and 'valid' padding	50 filters of size (3,3), used Relu activation function, pool_size=(4, 4) strides (1,1) and 'valid' padding	Maxpool layer	Yes	100 neurons	1 neuron	loss='binary_crossentropy', optimizer="sgd",	10	128
Accuracy on training set: 0.6634264884568651 %, Accuracy on validation set: 0.6572237960339944 %										

It is observed with many experiments that tuning of Hyperparameters in CNN is important so that desire model accuracy can be obtained. Deciding Hyperparameters value manually is tedious and time-consuming procedure. By increasing filter size no of trainable parameters are reduced but at the same time some information in the image is lost and the model has the problem of overfit.

Model tuning is the experimental process of finding the optimal values of hyperparameters to maximize model performance. Hyperparameters are the set of variables whose values cannot be estimated by the model from the training data. These values control the training process.

The Keras Tuner[9] is a library that helps you pick the optimal set of hyperparameters for your TensorFlow program. The process of selecting the right set of hyperparameters for your machine learning (ML) application is called hyperparameter tuning. For this emergency vehicle data set, it was observed that Model6 is giving optimum result.

V. CONCLUSIONS

Keras tuner is an open-source python library developed exclusively for tuning the hyperparameters of ANN. Using this library, selected hypermeters of ANN can be tuned. Whereas for tuning parameters of CNN, it is best practice to repeat the procedure of compiling, training and evaluating model manually. After repeating experiments with many data sets it is observed that there is no fix solution for hyperparameter tuning, but tuning need to be performed manually depending on data set and problem statement.

ACKNOWLEDGMENT

Authors wish to thank teachers of Department of Computer Science, Thakur College of Science & Commerce for their constant support and motivation. We also thank for allowing to use resources of Thakur College of Science and Commerce, Mumbai.

REFERENCES

[1] Amita Kapoor, Antonio Gulli , Sujit Pal Deep Learning with TensorFlow and Keras, Packt Publisher, Third Edition, 2022

[2] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. Large-Scale Evolution of Image Classifiers. In Proceedings of the 34th International Conference on Machine Learning - Volume 70, pages 2902–2911, Sydney, NSW, Australia, 2017

[3] Jie Fu, Hongyin Luo, Jiashi Feng, Kian Hsiang Low, and Tat-Seng Chua. Dr MAD: Distilling Reverse-Mode Automatic Differentiation for Optimizing Hyperparameters of Deep Neural Networks. In Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, pages 1469–1475, 1 2016.

[4] Ian Goodfellow, Yoshua Bengio, Aaron Courvile, Deep Learning, MIT Press, 2016

[5] Ilievski, I., Akhtar, T., Feng, J., & Shoemaker, C. (2017). Efficient Hyperparameter Optimization for Deep Learning Algorithms Using Deterministic RBF Surrogates. Proceedings of the AAAI Conference on Artificial Intelligence, 31(1).

[6] L. Wu, G. Perin and S. Picek, "I Choose You: Automated Hyperparameter Tuning for Deep Learning-based Side-channel Analysis," in *IEEE Transactions on Emerging Topics in Computing*, 2022, doi: 10.1109/TETC.2022.3218372.

[7] Steven R. Young, Derek C. Rose, Thomas P. Karnowski, Seung-Hwan Lim, and Robert M. Patton. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments - MLHPC '15, pages 1–5, Austin, Texas, USA, 2015. ACM Press.

[8] T. Triwiyanto, I. P. A. Pawana and M. H. Purnomo, "An Improved Performance of Deep Learning Based on Convolution Neural Network to Classify the Hand Motion by Evaluating Hyper Parameter," in IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 28, no. 7, pp. 1678-1688, July 2020, doi: 10.1109/TNSRE.2020.2999505. https://doi.org/10.1609/aaai.v31i1.10647

[9] Keras Tuner, https://keras.io/keras_tuner/ Accessed on 20th march 2023