8/11/2022

Optimizing Deep Learning Models

With Hyperparameter Tuning and Implementation of Transfer Learning

Tharun Prabhakar

40027784

Contents

[**Problem Statement** 2](#_Toc112227370)

[**About Datasets** 3](#_Toc112227371)

[**Description of CIFAR-10 Dataset:** 3](#_Toc112227372)

[**Description of Cats and Dogs Dataset:** 6](#_Toc112227373)

[**Algorithms Used** 8](#_Toc112227374)

[**Convolutional Neural Network:** 8](#_Toc112227375)

[**ResNet50:** 9](#_Toc112227376)

[**EfficientNetB0:** 10](#_Toc112227377)

[**Experiments and Implementation** 13](#_Toc112227378)

[**System Specifications where the Base Model was Developed:** 13](#_Toc112227379)

[**Versions of Packages Used:** 13](#_Toc112227380)

[**System Specifications for Optimisation of Base Model:** 15](#_Toc112227381)

[**Base Model Architecture:** 16](#_Toc112227382)

[**Optimisation of Base Model:** 16](#_Toc112227383)

[**Base Model Trained using EarlyStopping:** 16](#_Toc112227384)

[**Base model with Reducing Learning Rate:** 17](#_Toc112227385)

[**Base Model with Earlystopping and Learning Rate Scheduler:** 17](#_Toc112227386)

[**Base Model with Learning Rate Scheduler and Checkpoint:** 17](#_Toc112227387)

[**Base Model with Reduce Learning Rate:** 18](#_Toc112227388)

[**Base Model with Reduce Learning Rate and Checkpoints:** 18](#_Toc112227389)

[**Base Model with All 3 Callbacks:** 18](#_Toc112227390)

[**Hyperparameter Tuning of Base Model:** 18](#_Toc112227391)

[**Hyperparameters tuning to get kernel size:** 18](#_Toc112227392)

[**Hyperparameter tuning to get Filters, Kernel sizes and nodes of Hidden Layers:** 20](#_Toc112227393)

[**Hyperparameter Tuning to get Optimal Learning Rate:** 22](#_Toc112227394)

[**Retraining of the Base Model:** 23](#_Toc112227395)

[**Transfer Learning:** 23](#_Toc112227396)

[**User Interface:** 24](#_Toc112227397)

[**Testing** 26](#_Toc112227398)

[**Results** 29](#_Toc112227399)

[**Conclusion** 31](#_Toc112227400)

[**References** 32](#_Toc112227401)

# **Problem Statement**

We live in the era of data. With the Internet of Things (IoT) and Artificial Intelligence (AI) becoming ubiquitous technologies, we now have huge volumes of data being generated. Differing in form, data could be speech, text, image, or a mix of any of these. In the form of photos or videos, images make up for a significant share of global data creation.

Since the vast amount of image data, we obtain from cameras and sensors is unstructured, we depend on advanced techniques such as machine learning algorithms to analyse the images efficiently. Image classification is probably the most important part of digital image analysis. It uses AI-based deep learning models to analyse images with results that for specific tasks already surpass human-level accuracy (for example, in Face Recognition).

Image classification is the task of categorizing and assigning labels to groups of pixels or vectors within an image dependent on particular rules. The categorization law can be applied through one or multiple spectral or textural characterizations. This project involves the optimisation of image classification neural network that is been trained on CIFAR-10 dataset and evaluating the performance of model that has been retrained with specific set of images and implementation of transfer learning and its evaluation of its performance.

# **About Datasets**

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset contains labels of 10 classes Aeroplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship and Truck. These labels are numbered from 0 to 9. These classes are also mutually exclusive that is there is no link between automobile class and truck class they are treated as separate classes. Each image is blurry with of 32x32 dimension.

For the purpose of retraining the model cats and dog’s dataset were used so try to retrain the model with the images of cats and dogs so that the miss comings of the base model that was selected as the best model could be improved. The dataset consists of 25000 images of both cats and dogs. The images were of size 375x500 dimensions with ‘RGB’. Hence requirement resizing the image is required.

## **Description of CIFAR-10 Dataset:**

The dataset is default dataset present in TensorFlow from which it is imported.



Once called the dataset is loaded using load dataset feature in keras.

Graphical user interface

Description automatically generated with low confidence

From the above image we can see that the images and the labels are directly loaded to the training and testing set.

A screenshot of a computer

Description automatically generated with medium confidence

The images are then divided as 50000 for training set and 10000 for testing set. The labels are in the form of 2D array.

Text

Description automatically generated

This array needs to be converted to 1D so that it’s easy for matching or identifying of images. Reshape function is used to change the dimension of the labels.

Text

Description automatically generated

Graphical user interface

Description automatically generated

Sample image has been tested and its corresponding labels are checked. The labels are encoded so requires labelling them.



The training set contains 5000 images of each class summing up to 50000 images and the testing set contains 1000 images of each class for testing. These images are in random order in both the sets, so the training does not go in order of class labels.

Text

Description automatically generated with medium confidence

Text

Description automatically generated

The train and test data are then normalised by dividing the array with 255 so that the colours are reduced between the range of 0 and 1.



## **Description of Cats and Dogs Dataset:**

The datasets consist of 25000 images with 12500 for dogs and 12500 for cats.

A picture containing calendar

Description automatically generated

Once the dataset directory is accessed, the images have been separated into two variables cats and dogs

Graphical user interface, text

Description automatically generated

Number of images were checked and found to be 25000. Then labels were assigned as ‘3’ for cat and ‘5’ for dog which was previously defined by the CIFAR-10 dataset.

Text

Description automatically generated

Since the size of the images were more than the required for the model to retrain resizing of the images to dimensions 32x32 is done.

Text

Description automatically generated

After the image is resized, the data is then converted to array so that it can be interpreted by the algorithm. After the conversion is done, normalisation is done so that the ranges of RGB values are reduced and fall between the range of 0 to 1.

Graphical user interface, text, application

Description automatically generated

Train test split was imported from sklearn.model\_selection to split the data into train and test sets.

Text, letter

Description automatically generated

# **Algorithms Used**

## **Convolutional Neural Network:**

Convolutional neural network or CNN is the most popular neural network used in image classification problems. Convolutional neural networks are very good at picking up on patterns in the input image, such as lines, gradients, circles, or even eyes and faces. It is this property that makes convolutional neural networks so powerful for computer vision. Unlike earlier computer vision algorithms, convolutional neural networks can operate directly on a raw image and do not need any pre-processing. A convolutional neural network is a Feed Forward network, often with up to 20 or 30 layers. The power of a convolutional neural network comes from a special kind of layer called the convolutional layer. The usage of convolutional layers in a convolutional neural network mirrors the structure of the human visual cortex, where a series of layers process an incoming image and identify progressively more complex features. The CNN has 3 parts the convolution, classification, and output. It has 3 layers convolutional, pooling and fully connected layer. When a filter is applied matrix multiplication is done and result is updated to next filter. The filter moves block by block which is determined by stride. Default value of stride is 1. Padding is done so that the features are not lost on the feature map. Padding adds a layer of cells at the edges of the matrix generally the value inside cells be 0. Pooling layer is then used for dimensionality reduction. Generally max pooling method is used that selects the max value on the feature map. This is done so that chances of overfitting reduce, and time consumption of training is reduced. Before sending the data to fully connected layer flattening is done. Illustration of a basic CNN has been shown below.

**Diagram

Description automatically generated**

## **ResNet50:**

ResNet stands for Residual Network. ResNet has many variants that run on the same concept but have different numbers of layers. Resnet50 is used to denote the variant that can work with 50 neural network layers. When working with deep convolutional neural networks to solve a problem related to computer vision, we engage in stacking more layers. As the number of layers of the neural network increases, the accuracy levels may get saturated and slowly degrade after a point. As a result, the performance of the model deteriorates both on the training and testing data. This degradation is not a result of overfitting. Instead, it may result from the initialization of the network, optimization function, or, more importantly, the problem of vanishing or exploding gradients. ResNet was created with the aim of tackling this exact problem. Deep residual nets make use of residual blocks to improve the accuracy of the models. The concept of “skip connections,” which lies at the core of the residual blocks, is the strength of this type of neural network. These skip connections work in two ways. Firstly, they alleviate the issue of vanishing gradient by setting up an alternate shortcut for the gradient to pass through. In addition, they enable the model to learn an identity function. This ensures that the higher layers of the model do not perform any worse than the lower layers. In short, the residual blocks make it considerably easier for the layers to learn identity functions. As a result, ResNet improves the efficiency of deep neural networks with more neural layers while minimizing the percentage of errors. In other words, the skip connections add the outputs from previous layers to the outputs of stacked layers, making it possible to train much deeper networks than previously possible.

Diagram

Description automatically generated

## **EfficientNetB0:**

EfficientNet similar to ResNet was created to tackle vanishing gradient problem. It not only does depth scaling by adding more number of layers but also scales the width and resolution of the data also known as scaling using compound. The width here corresponds to the number of nodes which is high compared to ResNet to increase the number of feature maps. These contribute to feature called model scaling. Model scaling is the process of scaling up a base convolutional neural network to endow it with greater computational complexity and consequently more representational power. Model scaling approaches typically focus on maximizing accuracy. The conventional practice for model scaling is to arbitrarily increase the CNN depth or width, or to use larger input image resolution for training and evaluation. While these methods do improve accuracy, they usually require tedious manual tuning, and still often yield suboptimal performance. It’s typically performed with pretrained models and importing them. The compound scaling method is justified by the intuition that if the input image is bigger, then the network needs more layers to increase the receptive field and more channels to capture more fine-grained patterns on the bigger image. Generally, the models are made too wide, deep, or with a very high resolution. Increasing characteristics helps the model initially but it quickly saturates, and the model made just has more parameters and is therefore not efficient. In EfficientNet they are scaled in a more principled way i.e., gradually everything is increased. The base EfficientNet-B0 network is based on the inverted bottleneck residual blocks of MobileNetV2, in addition to squeeze-and-excitation blocks.

The first thing is any network is its stem after which all the experimenting with the architecture starts which is common in all the eight models and the final layers.

Diagram

Description automatically generated

After this each of them contains 7 blocks. These blocks further have a varying number of sub-blocks whose number is increased as we move from EfficientNetB0 to EfficientNetB7.

EfficientNetB0 has 237 layers, but all these layers are made from 5 modules that is shown below and stem that is shown above.

Diagram

Description automatically generated

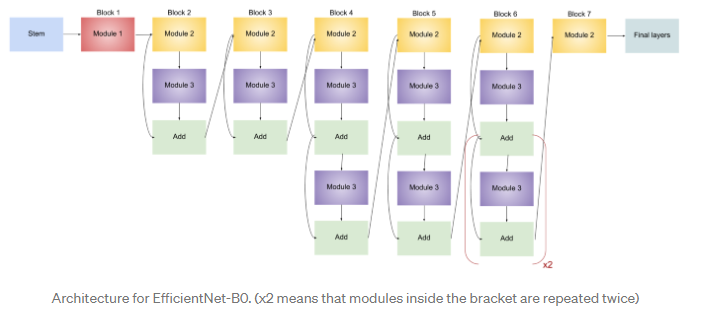
* Module 1 — This is used as a starting point for the sub-blocks.
* Module 2 — This is used as a starting point for the first sub-block of all the 7 main blocks except the 1st one.
* Module 3 — This is connected as a skip connection to all the sub-blocks.
* Module 4 — This is used for combining the skip connection in the first sub-blocks.
* Module 5 — Each sub-block is connected to its previous sub-block in a skip connection, and they are combined using this module.

These modules are further combined to form sub-blocks which will be used in a certain way in the blocks.

A picture containing text, screenshot, businesscard

Description automatically generated

* **Sub-block 1**— This is used only used as the first sub-block in the first block.
* **Sub-block 2** — This is used as the first sub-block in all the other blocks.
* **Sub-block 3**— This is used for any sub-block except the first one in all the blocks.



The table shown below denotes the kernel size for convolution operations along with the resolution, channels, and layers in EfficientNet-B0.

Table

Description automatically generated

# **Experiments and Implementation**

## **System Specifications where the Base Model was Developed:**

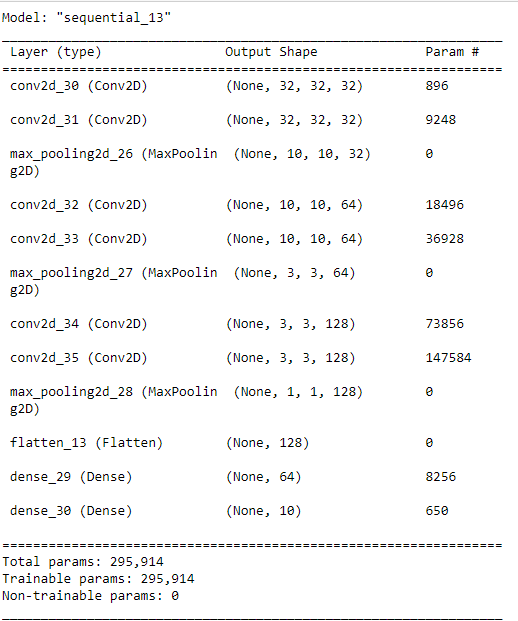
|  |  |
| --- | --- |
| **Processor** | Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz 3.00 GHz |
| **Installed RAM** | 16.0 GB (15.8 GB usable) |
| **System type** | 64-bit operating system, x64-based processor |

## **Versions of Packages Used:**

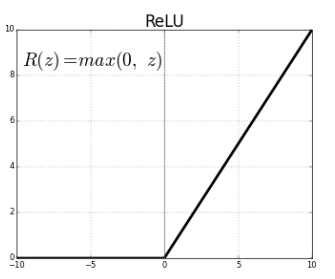
|  |  |  |
| --- | --- | --- |
| Package |  | Version |
| Pandas |  | 1.4.3 |
| NumPy |  | 1.22.4 |
| Seaborn |  | 0.11.2 |
| TensorFlow |  | 2.9.1 |
| Sklearn |  | 1.1.1 |
| PIL |  | 9.0.1 |

**Base Model:**

The base model that had been selected as the best model from previous research has been selected for optimisation to improve its accuracy and to experiment with it. The model is a 6-layer convolutional network with 3 pooling layers arranged as 2 convolutional layer and 1 pooling layer. The first two convolutional layers have 32 filters with kernel size of 3x3 and ‘ReLu’ activation. The kernel initializer used is ‘he\_uniform’ and padding as ‘same’. The next of convolutional layers have 64 filters with 3x3 dimensions and ‘ReLu’ activation. The final set has two convolutional layers with 128 filters with 3x3 dimensions with same configuration as the previous layers. The pooling layer between the sets of convolutional layers is of dimensions 3x3. These convolutional layers are followed by flatten layer which is followed by a hidden layer with 64 nodes and output layer with 10 nodes to predict the class labels with ‘softmax’ activation function. The model is compiled with ‘adam’ optimizer, ‘sparse\_categorical\_crossentropy’ as loss function and metric to monitor as ‘accuracy’. The model was given a batch size of 64 and was trained for 10 epochs.



**Rectified Linear Unit** Activation Function or ReLu is one of the most used activation functions



It gives the output as 0 if the value is equal to or less than 0 and gives z if the value is equal to z. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

**Softmax** is a function that turns a vector of K real values into a vector of K real values that sum to 1. The input values can be positive, negative, zero, or greater than one, but the softmax transforms them into values between 0 and 1, so that they can be interpreted as probabilities. If one of the inputs is small or negative, the softmax turns it into a small probability, and if an input is large, then it turns it into a large probability, but it will always remain between 0 and 1.

**Sparse Categorical Crossentropy** is an error in classification computed for the whole training set. It is similar Categorical Crossentropy only difference is that it’s used to when the data is integer labelled rather than one-hot encoding.

**Epochs** means number of times the models have to be trained to get better required accuracy.

**Adaptive Movement Estimation or Adam algorithm** is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing. It is very efficient when there is huge data to be trained upon. It requires less memory and is efficient.

**Kernel initializer** is used to set the value of the filter kernel with which feature extraction takes places. The option used here ‘he\_uniform’. This function is used to assign values from uniform distribution table to the kernel.

**Padding** is used to add a layer around the feature map so that there is no feature loss. The option used here is ‘same’. This option is used to preserve the dimensions of the feature map so that the output matches the input.

## **System Specifications for Optimisation of Base Model:**

|  |  |
| --- | --- |
| **Processor** | AMD Ryzen 3750H |
| **Ram** | 16.0 GB (15.8 GB usable) |
| **System Type** | 64-bit operating system, x64-based processor |
| **Graphic Card** | Nvidia RTX 2060 Super |

Processing of the model was performed using GPU instead of CPU to Fasten the process of training.

tensorflow-gpu package and CUDA toolkit was installed to access the GPU and following code was used to access the GPU.

Graphical user interface, text, application, email

Description automatically generated

## **Base Model Architecture:**

Table

Description automatically generated with low confidence

## **Optimisation of Base Model:**

### **Base Model Trained using EarlyStopping:**

The Base model that built and given a callback feature of Earlystopping from tensorflow.keras.callbacks. The purpose of using earlystopping is that the model stops training if there is no improvement in metric that is being monitored. The validation loss was monitored in this model using earlystop and patience of 2 was given. Patience is used so that the callback checks the monitored value after every 2 epochs. The model was fit and made to train for 15 epochs, but the model trained for 7 epochs and stopped because of earlystopping being used.



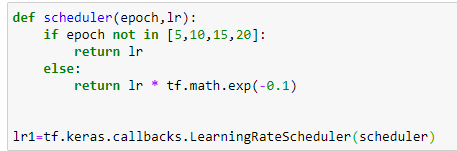
### **Base model with Reducing Learning Rate:**

The base model was built and Reduce learning rate feature was used from tensorflow.keras.callback. This feature reduces the learning rate if the monitored metrics is not improved. A patience of 3 was given which means after every 3 epochs the monitored value that is validation accuracy is checked and if there is no improvement in it then the learning rate gets reduced. The model is trained for 10 epochs.



### **Base Model with Earlystopping and Learning Rate Scheduler:**

The next model uses a learning scheduler from tensorflow.keras.callbacks. The learning scheduler modifies the learning rate as per the requirement. In this model scheduler reduces the learning rate after every 5 epochs.



A function is defined to reduce the learning rate after every 5 epochs by multiplying it exponent of -0.1 and is given to the function. Along with learning rate scheduler, earlystopping is used to prevent overtraining the model. The model is compiled and is made to run for 20 Epochs.

### **Base Model with Learning Rate Scheduler and Checkpoint:**

The previous model that was built had better accuracy at intermediate epochs to fetch the best result. tensorflow.keras.callbacks.ModelCheckpoint is used to create checkpoints when model provide high accuracy score. The checkpoint has weights of the model when it had the highest accuracy.



### **Base Model with Reduce Learning Rate:**

The model is fit reduce learning rate with a patience of 2. This means the model’s accuracy of the validation set is checked every 2 epochs and if there’s no difference the learning rate is reduced.



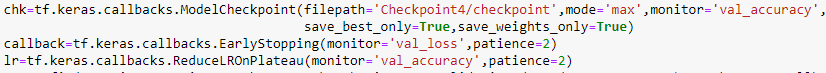
### **Base Model with Reduce Learning Rate and Checkpoints:**

The model is fit with both the callbacks to get the best weights for the model to be stored in the checkpoint and retrieved. The model is the fit for 20 epochs.



### **Base Model with All 3 Callbacks:**

The model is now fit with all 3 Callbacks of earlystopping, reduce learning rate and Checkpoint to check if the validation accuracy can be increased.



## **Hyperparameter Tuning of Base Model:**

### **Hyperparameters tuning to get kernel size:**

A function is defined to compile a model with architecture similar to base model, but number of nodes at hidden layer, kernel size of the filters and pooling layer’s kernel size is going to be decided by the RandomSearch from keras\_tuner package.



**Graphical user interface, text, application

Description automatically generated**

Tuner is set get model with features that provide the highest accuracy and maximum trial of 3 is given to find the best features where the model is trained of 3 epochs per trial with validation split of 20%.

After validation the model with best result was loaded. The summary of the model is shown below.

Table

Description automatically generated

As seen from the summary the nodes in the hidden layer have been changed to 80 as provided by the tuner. The model is then fit for 10 epochs.

### **Hyperparameter tuning to get Filters, Kernel sizes and nodes of Hidden Layers:**

The tuner was made to find the number of filters in each convolutional layer and kernel size of the filters, kernel size of the max pooling layer and number of nodes in hidden layers. The tuner is done for 3 trials and 3 epochs per trial and model with highest validation accuracy is taken.

Text

Description automatically generated

The model is loaded with weights that provided best validation accuracy and is trained for 10 epochs and checkpoint to store the model with highest validation score.

Table

Description automatically generated

The model had 32, 128, 144, 224, 96, 176 filters in each convolutional layer and hidden layer has been set to 80 nodes.

### **Hyperparameter Tuning to get Optimal Learning Rate:**

The tuner is set to select the optimal learning rate from the given list of learning rate to improve the accuracy of the model.

Graphical user interface, text, application

Description automatically generated

The tuner is set to run for 7 trials and the model is run for 10 epochs per trial to get the optimal score. The model is then fit with the best weights.

Table

Description automatically generated with medium confidence

## **Retraining of the Base Model:**

The base model was retrained with the images of dogs and cats as the model was struggling to get the difference between them. The dataset was processed and was split into test and train dataset using train\_test\_split with 25% of images reserved for the test set.

1. **Retraining the model with 10 Epochs**
2. **Retraining the model with 5 epochs**
3. **Retraining the model with 3 epochs**

The model was retrained with same structure and optimisers, only difference being the change in epochs.

Table

Description automatically generated with low confidence

## **Transfer Learning:**

Transfer learning, used in machine learning and deep learning, is the use of pretrained model on a new model. In transfer learning, a machine exploits the knowledge gained from a previous task to improve generalization about another.

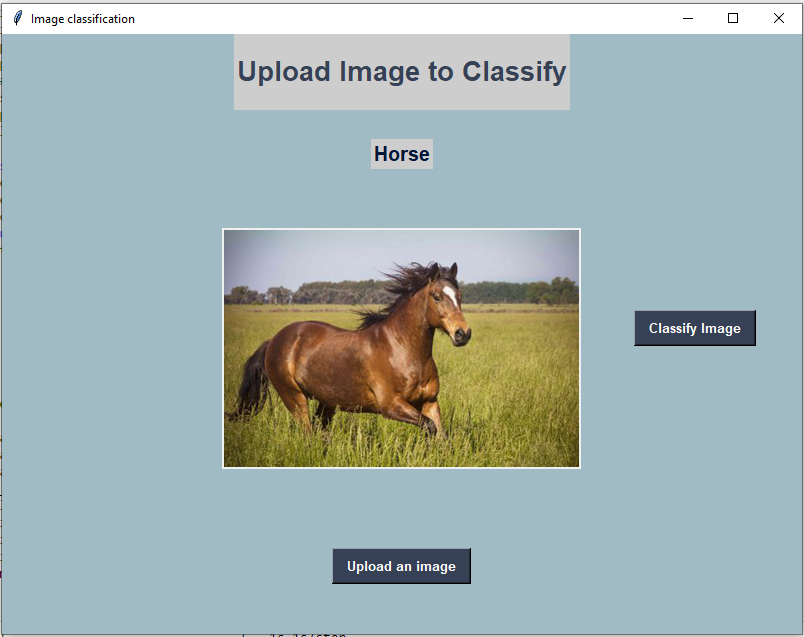
Pre-trained Resnet50 has been used instead of base model’s convolutional layers to extract the features from the image and is provided to the fully connected layers. The weights used were derived from ‘ImageNet’. Two models were developed one with 64 nodes in the hidden layer and another with 80 nodes in the hidden layer. Next with same number hidden layers EfficientNet models were built and were made to run for 15 and 20 epochs

Text

Description automatically generated

## **User Interface:**

A User Interface was developed that allows the user to upload the image from the local folder and allows the user to check the class of the image or object or animal present in the image



Graphical user interface, text

Description automatically generated

A horse running in a field

Description automatically generated with low confidence

# **Testing**

The models that were built were tested using the given set of images belonging to different classes.

A picture containing text, outdoor, boat

Description automatically generated

A picture containing cat, sitting, white, mammal

Description automatically generated

A collage of a cat

Description automatically generated with medium confidence

A picture containing text, phasianid

Description automatically generatedA collage of a dog

Description automatically generated with medium confidence

A picture containing text, aircraft, airplane

Description automatically generated

A picture containing text, transport, watercraft, boat

Description automatically generated

A large cruise ship

Description automatically generated with low confidence

The images are of varied size and clarity as shown above. The models were made to predict the class of these images and the score are shown below.

Graphical user interface

Description automatically generated with medium confidence

# **Results**

The model is fit with training set and the result predictions for the various models is obtained and accuracy score is checked.

**Accuracy Score** is a metric that is used for checking the performance of the model of classification problem. It compares with predicted value with test value and gives the score that ranges between 0 and 1. A model with high accuracy is a good model. It is ratio of true predicted values to all predicted values.

Table

Description automatically generated

1. All the models that were built were overfitting as there is huge difference between test and train sets accuracy
2. As the model being introduced with callbacks the accuracy of the model is increased indicating that the callbacks prevent the overfitting of the model and provides efficient training of the model.
3. The use of ReduceLRonPlateau with less patience the model learns better with less epochs.
4. EarlyStopping stops the model’s overfitting.
5. Checkpoints are an efficient way to store and load model weights to achieve better accuracy if the model had varying learning.
6. The Usage of GPU has improved the training time and results of the models.
7. Hypertuning of the models with kernel size, number of filters and number of nodes has increased the accuracy of the model.
8. The models that were retrained using the datasets of cats and dogs with the base model’s architecture has made the model to become narrow resulting in poor accuracy of the model. As a result, the models were only predicting the images of cats and dogs well and misclassifying other images.
9. Transfer Learning is an efficient way to train and develop models.
10. With the increase in the number of layers the accuracy of the models increases.
11. With Increase in number of epochs the models have trained better to produce a better accuracy score than base model.
12. The accuracy of transfer learning algorithm has been affected by the presence of only one hidden layer. The models would have performed better with introduction of more hidden layer and nodes.
13. The Scores of the models indicate that all the models were able to predict more that 50% of the total images that has been provided to the model.
14. The Base CNN, Hypertuned model with filter and Kernel size and EfficientNet model were able to predict 18 images out of 22. The retrained models were only able to predict cats or dogs and no other classes

# **Conclusion**

Thus, we can conclude that the model’s performance can be improved with hyperparameter tuning and with the usage of callbacks from TensorFlow package. Retraining of models makes the model become more narrowed. Usage of GPU has major influence in not only reducing the training time but also improves the mathematical calculations resulting in better accuracy. Transfer learning improves the model’s accuracy by adding more layers to the model with getting affected by vanishing gradient problem. Accuracy of those models can also be improved my adding more hidden layers. Using Model Scaling and a larger dataset one might just get a perfect accuracy model for classification of images.

We were also able to create a code to test all of our models and various images off the internet and give near to accurate assessment. Based on the testing process and accuracy score, we can select the EfficientNet model as the best model. A simple User Interface was developed that uploads the image and predicts the output.

# **References**

1. Dataset Download: <https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz>
2. [cifar10  |  TensorFlow Datasets](https://www.tensorflow.org/datasets/catalog/cifar10)
3. [Cats-vs-Dogs | Kaggle](https://www.kaggle.com/datasets/shaunthesheep/microsoft-catsvsdogs-dataset)
4. [Basic Introduction to Feed-Forward Network in Deep Learning (analyticsvidhya.com)](https://www.analyticsvidhya.com/blog/2022/03/basic-introduction-to-feed-forward-network-in-deep-learning/)
5. [Activation Functions in Neural Networks | by SAGAR SHARMA | Towards Data Science](https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6)
6. [Implementing feedforward neural networks with Keras and TensorFlow - PyImageSearch](https://pyimagesearch.com/2021/05/06/implementing-feedforward-neural-networks-with-keras-and-tensorflow/)
7. [An intuitive guide to Convolutional Neural Networks (freecodecamp.org)](https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/)
8. [Convolutional Neural Network Definition | DeepAI](https://deepai.org/machine-learning-glossary-and-terms/convolutional-neural-network)
9. [What Is Transfer Learning? A Simple Guide | Built In](https://builtin.com/data-science/transfer-learning)
10. [🚗 🦅⛵Cifar-10 Object Recognition ResNet Acc~94% | Kaggle](https://www.kaggle.com/code/datajameson/cifar-10-object-recognition-resnet-acc-94)
11. [ResNet-50 neural network architecture [56]. | Download Scientific Diagram (researchgate.net)](https://www.researchgate.net/figure/ResNet-50-neural-network-architecture-56_fig4_336805103)
12. [Deep Residual Networks (ResNet, ResNet50) 2022 Guide - viso.ai](https://viso.ai/deep-learning/resnet-residual-neural-network/)
13. [Complete Architectural Details of all EfficientNet Models | by Vardan Agarwal | Towards Data Science](https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142)
14. [EfficientNet: Scaling of Convolutional Neural Networks done right | by Armughan Shahid | Towards Data Science](https://towardsdatascience.com/efficientnet-scaling-of-convolutional-neural-networks-done-right-3fde32aef8ff)
15. [tf.keras.applications.efficientnet.EfficientNetB0  |  TensorFlow Core v2.9.1](https://www.tensorflow.org/api_docs/python/tf/keras/applications/efficientnet/EfficientNetB0)