

Project Report

On

Image Classification on Intel Dataset using
CNN and Transfer Learning



Submitted in partial fulfilment for the award of
Post Graduate Diploma in Big Data Analytics (PGDBDA)
From Know IT(Pune)

Guided by:

Mrs. Trupti Joshi & Mr. Prasad Deshmukh

Submitted By:

Ashlesha Purandare (230943025008)

Mangesh Jadhav (230943025027)

Swarada Pathak (230943025036)

Sahil Kale (230943025042)

CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

This is to certify that

Ashlesha Purandare (230943025008)

Mangesh Jadhav (230943025027)

Swarada Pathak (230943025036)

Sahil Kale (230943025042)

Have successfully completed their project on

Image Classification on Intel Dataset using
CNN and Transfer Learning

Under the guidance of Mrs. Trupti Joshi and Prasad Deshmukh Sir

ACKNOWLEDGEMENT

This project Image Classification on Intel Dataset using CNN and Transfer Learning was a great learning experience for us and we are submitting this work to CDAC Know IT (Pune).

We all are very glad to mention the name Mrs. Trupti Joshi and Mr. Prasad Deshmukh for their valuable guidance to work on this project. His guidance and support helped us to overcome various obstacles and intricacies during the course of project work.

We are highly grateful to Mr. Vaibhav Inamdar Manager (Know IT), CDAC, for his guidance and support whenever necessary while doing this course Post Graduate Diploma in Big Data Analytics (PGDBDA) through CDAC ACTS, Pune.

Our most heartfelt thanks go to Mrs. Bakul Joshi (Course Coordinator, PGDBDA) who gave all the required support and kind coordination to provide all the necessities like required hardware, internet facility and extra Lab hours to complete the project and throughout the course up to the last day here in CDAC Know IT, Pune.

TABLE OF CONTENTS

ABSTRACT

1. INTRODUCTION
2. DATA COLLECTION AND FEATURES
3. SYSTEM REQUIREMENTS
 - 3.1 Software Requirements
 - 3.2 Hardware Requirements
4. FUNCTIONAL REQUIREMENTS
5. ARCHITECTURE
6. DEEP LEARNING ALGORITHMS
7. TRANSFER LEARNING MODELS
 - 7.1 VGG 16
 - 7.2 MobileNet V2
 - 7.3 Inception V3
8. GRADIO
9. CONCLUSION AND FUTURE SCOPE
10. REFERENCES

ABSTRACT

Image classification is a fundamental task in computer vision with wide variety of applications. The goal of this project is to classify images from the Intel Dataset using Convolutional Neural Networks (CNNs) and transfer learning. The dataset contains a diverse set of natural scene images across six categories: buildings, forests, glaciers, mountains, sea, and streets. The methodology involves data preprocessing, CNN model design and training, evaluation and implementing Transfer Learning models and compute accuracy metrics on Intel dataset. With the help of deep learning, our model aims to accurately detect and classify various types of natural scene images. The developed solution will assist in environmental monitoring such as monitoring deforestation, tracking changes in Glaciers, urban planning, and effectively contribute to environmental conservation.

1. INTRODUCTION

Images are a common source of information in a variety of areas in the age of ever growing digital content. The ubiquity of social media, e-commerce, and the Internet of Things (IoT) has made it critical to analyse and categorize photos in an efficient manner. One of the core tasks in computer vision is image classification, which is labelling or classifying images according to their visual content. Convolutional Neural Networks (CNNs) have shown impressive performance in a wide range of applications, making them an effective tool for image classification.

Using the Intel picture Classification dataset, this paper investigates the use of CNNs and transfer learning methods in picture classification. The Intel Image Classification dataset provides a varied range of natural settings that includes forests, buildings, and landscapes.

Objectives:

The primary objectives of this report are:

- To investigate the effectiveness of Convolutional Neural Networks (CNNs) in image classification tasks.
- To explore the application of transfer learning, a technique that leverages pre-trained CNN models, in image classification on the Intel dataset.
- To evaluate the performance of different CNN architectures and transfer learning strategies in classifying images from the Intel dataset.

2. Dataset Collection and Features

Data Sources

For our project, we utilized the data on Kaggle which is a public platform to access varieties of datasets. The data published by Intel on natural images in six varieties of classification. The images are a collection of Natural scenes from around the world. The dataset consists of three divisions: train, test, and prediction. This dataset is commonly used for image classification tasks, such as training machine learning models, that of CNN (Convolution Neural Network) and Transfer Learning to recognize different types of natural scenes.

Dataset Size

This dataset contains around 24,3355 images of natural scenes from around the world.

The images are distributed among six categories:

- Buildings
- Forest
- Glacier
- Mountain
- Sea
- Street

Each image is resized to 150x150 pixels.

The dataset is split into three parts: train, test, and prediction.

- Train: 14,034 images
- Test: 3,000 images
- Prediction: 7,301 images

Features/Attributes:

The overview of key features (attributes) within our dataset:

1. Streets:

Attributes:

The streets could consist of objects like vehicles, poles, dividers, humans and so on. This is to depict the streets accurately. Dominant objects: Roads, pavements, sidewalks, buildings, vehicles, traffic lights, signs.

Background: Might include sky, trees, pedestrians, or other urban elements.

2. Glaciers:

Attributes:

Glacier images are seemingly blue and white in colour to depict the Glacial habitat.

Dominant object: White or blueish ice formations, crevasses, cracks, textures on the ice surface.

Background: Might include mountains, sky, water, or rocks.

3. Forests:

Attributes:

Forests majorly consist of flora and natural landscape.

Dominant objects: Trees of various sizes and species, foliage, branches, trunks.

Background: Might include sky, ground vegetation, rocks, or hills.

4. Buildings:

Attributes: Buildings largely consist of rectangular objects of varying heights.

Dominant objects: Rectangular shapes, windows, doorways, roofs, walls, textures like brick or concrete.

Background: Might include sky, roads, trees, or other buildings depending on the scene.

5. Mountains:

Attributes: The main attributes of mountains are triangular objects with peaks and slopes.

Dominant objects: Peaks, ridges, slopes, rocky formations, snow cover (depending on the season).

Background: Might include sky, forests, glaciers, or valleys.

6. Sea:

Attributes: These consist of large water bodies dominated by blue-green colours.

Dominant object: large body of water, waves, reflections, varying shades of blue or green depending on depth and conditions.

Background: Might include sky, coastline, ships, or clouds.

3. SYSTEM REQUIREMENTS

Hardware Requirements

Computer: A computer with sufficient processing power and memory to run data processing and analysis tasks. A modern multicore processor and at least 8 GB of RAM are recommended.

Storage: Adequate storage space to store the generated dataset and any additional datasets if required. An SSD (Solid State Drive) is recommended for faster data access.

Internet Connection: A stable internet connection for downloading and installing software packages and libraries, as well as for any online resources needed during the project.

Software Requirements

1. Operating System: Windows 10 or higher
2. Python: The project heavily relies on Python for data generation, analysis, and machine learning. Ensure Python is installed on your system.
3. Python Libraries: Install the following Python libraries and dependencies using package managers like pip or conda:
 - NumPy: For numerical computing.
 - Pandas: For data manipulation and analysis.
 - scikitlearn: For machine learning tasks.
 - Matplotlib and Seaborn: For data visualization.
 - TensorFlow: For numerical computation and large-scale machine learning.
 - Keras: Designed to provide a user-friendly and efficient interface for building and training neural networks.
 - Gradio: Build web-based user interfaces (UIs) for your machine learning models, APIs, or any arbitrary
 - Python function.

4.FUNCTIONAL REQUIREMENTS

Python 3:

- Python is a general purpose and high-level programming language.
- It is use for developing desktop GUI applications, websites and web applications.
- Python allows to focus on core functionality of the application by taking care of common programming tasks.
- Python is derived from many other languages, including ABC, Modula3, C, C++, Algol68, Small Talk, and Unix shell and other scripting languages.

5.ARCHITECTURE:

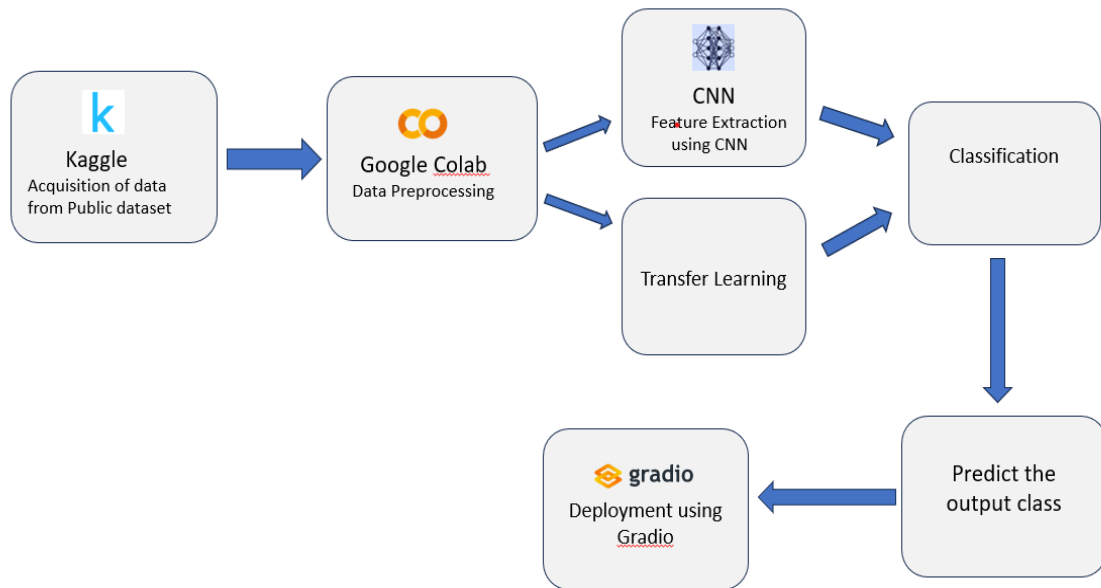


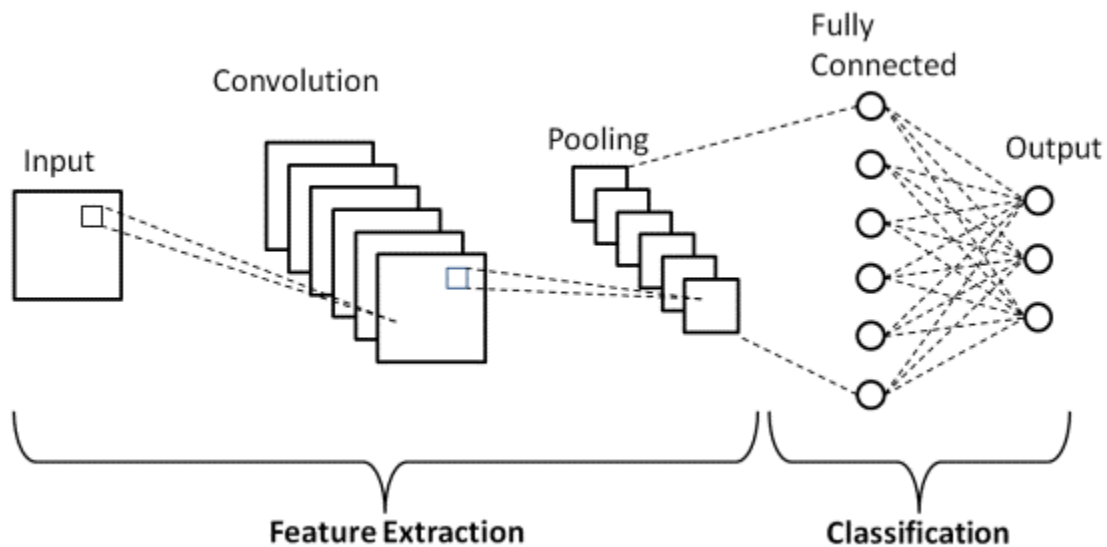
Fig: System Architecture of Intel Image Processing using CNN and transfer Learning

6. DEEP LEARNING ALGORITHMS

Basic CNN:

Algorithm Overview:

Convolutional neural networks (CNN) is a powerful deep learning algorithm capable of dealing with millions of parameters and saving the computational cost by inputting a 2D image and convolving it with filters/kernel and producing output volumes.



CNN layers

The CNN architecture consists of a number of layers (multi-building blocks)

1. Convolutional Layer:

In CNN architecture, the most significant component is the convolutional layer. It consists of a collection of convolutional filters (kernels). The input image, expressed as N-dimensional metrics, is convolved with these filters to generate the output feature map.

Kernel: A grid of discrete numbers or values describes the kernel. Each value is called the kernel weight. Random numbers are assigned to act as the weights of the kernel at the beginning of the CNN training process. Thus, the kernel learns to extract significant features.

Convolutional Operation: Initially, the CNN input format is described. The vector format is the input of the traditional neural network, while the multi-channelled image is the input of the CNN. RGB image format is three-channelled.

2. Pooling Layer:

Pooling layer in a Convolutional Neural Network (CNN) is a critical component that reduces the spatial dimensions (i.e., width and height) of the input volume for the next convolutional layer. It does this while retaining the most important information. Pooling helps in decreasing the computational power required to process the data through dimensionality reduction. Furthermore, it also helps in extracting dominant features which are rotational and positional invariant, thus aiding in the process of classification.

There are several types of pooling, but the two most common are:

- **Max Pooling:** In this pooling, where the maximum element is selected from the region of the feature map covered by the filter. By doing so, it captures the most prominent feature in the particular patch of the input.

3. Activation Layer:

An activation layer in a CNN is a layer that serves as a non-linear transformation on the output of the convolutional layer. It is a primary component of the network, allowing it to learn complex relationships between the input and output data.

The activation layer can be thought of as a function that takes the output of the convolutional layer and maps it to a different set of values. This enables the network to learn more complex patterns in the data and generalize better.

ReLU is the most commonly used activation function in most convolutional networks. It is a non-linear transformation that outputs 0 for all negative values and the same value as the input for all positive values. This allows the network to imbibe more complex patterns in the data.

Formula:

$$f(x) = \max(0, x)$$

Range: $[0, \infty)$

Sigmoid is another commonly used activation function, which outputs values between 0 and 1 for any given input. This helps the network to understand complex relationships between the input and output data but is more computationally expensive than ReLU.

Formula:

$$f(x) = 1 / (1 + \exp(-x))$$

Range: $(0,1)$

Tanh is the least commonly used activation function, which outputs values between -1 and 1 for any given input.

SoftMax: Used in the output layer of a neural network model for multi-class classification tasks. It converts the output scores from neurons into probabilities by taking the exponential of each output and then normalizing these values by dividing by the sum of all the exponentials.

Optimizers:

Optimizers are not directly applied to activation layers in Convolutional Neural Networks (CNNs).

They operate on the weights and biases of the network during the training process.

Here's a breakdown of the roles of optimizers and activation layers in CNNs:

Used to update the weights and biases of the network based on the calculated loss during training.

They adjust these parameters in a specific direction to minimize the loss function and improve the network's performance.

Common optimizers for CNNs include Adam and RMSprop.

a) Adam:

Combines the strengths of both momentum and adaptive learning rate concepts.

Momentum: Accumulates past gradients to smooth out updates and avoid getting stuck in local minima.

Adaptive learning rate: Adjusts the learning rate for each parameter individually based on its historical gradients.

Advantages:

- Often converges faster than SGD, especially for noisy or non-convex loss functions.
- Less sensitive to hyperparameter tuning compared to SGD.

Disadvantages:

- May be computationally expensive due to additional calculations for momentum and adaptive learning rates.
- Might not always be the best choice for very large datasets or specific problem domains.

b) RMSprop:

Addresses the issue of vanishing gradients in SGD, which can slow down learning in some scenarios.

Maintains a running average of squared gradients for each parameter.

Divides the learning rate by the square root of the average squared gradient to prevent large updates for parameters with consistently large gradients.

Advantages:

- Effective in dealing with vanishing gradients, especially for recurrent neural networks (RNNs) and LSTMs.
- Often converges faster than SGD in certain situations.

Disadvantages:

- Can be sensitive to the choice of learning rate and other hyperparameters.
- Might not be as efficient as Adam in some cases.

4. Fully Connected Layers:

The Fully Connected (FC) layer is a critical component of Convolutional Neural Networks (CNNs) and many other types of neural networks. Positioned after the convolutional and pooling layers in a CNN, the FC layer plays a pivotal role in the network's ability to make predictions and classifications based on the extracted features.

The fully connected layer integrates various features extracted in the previous convolutional and pooling layers and maps them to specific classes or outcomes. Each input from the previous layer connects to each activation unit in the fully connected layer, enabling the CNN to simultaneously consider all features when making a final classification decision.

Not all layers in a CNN are fully connected. Because fully connected layers have many parameters, applying this approach throughout the entire network would create unnecessary density, increase the risk of overfitting and make the network very expensive to train in terms of memory and compute.

5. Dropout:

Usually, when all the features are connected to the FC layer, it can cause overfitting in the training dataset. Overfitting occurs when a particular model works so well on the training data causing a negative impact in the model's performance when used on a new data. To overcome this problem, a dropout layer is utilised wherein a few neurons are dropped from the neural network during training process resulting in reduced size of the model. On passing a dropout of 0.3/ 30% of the nodes are dropped out randomly from the neural network.

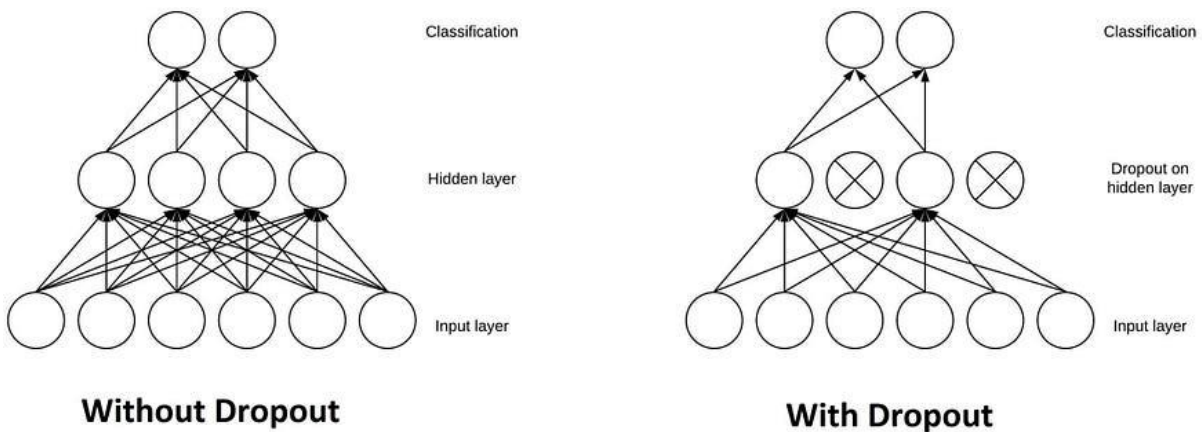


Fig: With- without Dropout

Output Layer:

The Output Layer is the final layer in a neural network architecture and is responsible for producing the model's output, making it crucial for achieving the specific objectives of a given task, such as classification, regression, or any other predictive modelling. This layer takes the processed information from the previous layers (which have gone through various transformations via convolutional, pooling, and fully connected layers, among others) and converts it into a form that matches the desired output format of the problem being solved.

Benefits:

Efficient feature extraction: CNNs are designed to automatically learn relevant features from images through convolutional layers. This eliminates the need for manual feature engineering, which can be time-consuming, subjective, and often ineffective. CNNs can automatically learn to identify patterns like shapes, textures, and colours that differentiate between different scene categories (buildings, forests, glaciers, etc.).

Superior performance: Compared to traditional machine learning algorithms, CNNs have consistently demonstrated superior performance on image classification tasks. In the Intel dataset, CNNs can achieve higher accuracy in classifying images into the correct categories compared to other methods.

Scalability: CNNs are well-suited for handling large datasets like the Intel Image Classification dataset, which contains thousands of images. As datasets grow larger, CNNs can continue to improve their performance.

Plot:

Model 1: Working with 10 Epochs, 1 layer each, training accuracy of 89.57%.

```
# fitting the model
trained_model1 = model1.fit(df_train, validation_data = df_test, batch_size=32, validation_split=0.3, epochs =10)

Epoch 1/10
439/439 [=====] - 28s 55ms/step - loss: 37.6404 - accuracy: 0.4220 - val_loss: 1.3416 - val_accuracy: 0.4653
Epoch 2/10
439/439 [=====] - 23s 52ms/step - loss: 0.9594 - accuracy: 0.6250 - val_loss: 1.4248 - val_accuracy: 0.4983
Epoch 3/10
439/439 [=====] - 23s 52ms/step - loss: 0.6554 - accuracy: 0.7493 - val_loss: 2.0097 - val_accuracy: 0.4963
Epoch 4/10
439/439 [=====] - 23s 52ms/step - loss: 0.5101 - accuracy: 0.8140 - val_loss: 2.3544 - val_accuracy: 0.4920
Epoch 5/10
439/439 [=====] - 23s 53ms/step - loss: 0.5260 - accuracy: 0.8218 - val_loss: 2.4716 - val_accuracy: 0.4867
Epoch 6/10
439/439 [=====] - 23s 53ms/step - loss: 0.4027 - accuracy: 0.8572 - val_loss: 2.9661 - val_accuracy: 0.4993
Epoch 7/10
439/439 [=====] - 24s 53ms/step - loss: 0.3759 - accuracy: 0.8733 - val_loss: 3.3719 - val_accuracy: 0.4837
Epoch 8/10
439/439 [=====] - 23s 53ms/step - loss: 0.3349 - accuracy: 0.8814 - val_loss: 3.6522 - val_accuracy: 0.4950
Epoch 9/10
439/439 [=====] - 24s 54ms/step - loss: 0.3305 - accuracy: 0.8871 - val_loss: 3.6317 - val_accuracy: 0.4867
Epoch 10/10
439/439 [=====] - 24s 55ms/step - loss: 0.3099 - accuracy: 0.8957 - val_loss: 4.0633 - val_accuracy: 0.4620
```

Result: The first CNN model shows testing accuracy of **46.20%**

Model 2: Added some more layers 10 Epoch, with training accuracy of 92.01%.

```
# fitting the model
trained_model2 = model2.fit(df_train, validation_data = df_test, batch_size=32, validation_split=0.3, epochs =10)

Epoch 1/10
439/439 [=====] - 145s 306ms/step - loss: 14.9077 - accuracy: 0.3127 - val_loss: 1.3597 - val_accuracy: 0.3997
Epoch 2/10
439/439 [=====] - 132s 300ms/step - loss: 1.1205 - accuracy: 0.5214 - val_loss: 1.2938 - val_accuracy: 0.4883
Epoch 3/10
439/439 [=====] - 131s 299ms/step - loss: 0.7887 - accuracy: 0.6699 - val_loss: 1.6385 - val_accuracy: 0.4697
Epoch 4/10
439/439 [=====] - 131s 299ms/step - loss: 0.5376 - accuracy: 0.7754 - val_loss: 1.8288 - val_accuracy: 0.4747
Epoch 5/10
439/439 [=====] - 131s 299ms/step - loss: 0.4170 - accuracy: 0.8325 - val_loss: 2.0417 - val_accuracy: 0.4787
Epoch 6/10
439/439 [=====] - 131s 298ms/step - loss: 0.3397 - accuracy: 0.8658 - val_loss: 2.4088 - val_accuracy: 0.4723
Epoch 7/10
439/439 [=====] - 131s 299ms/step - loss: 0.2786 - accuracy: 0.8908 - val_loss: 2.8098 - val_accuracy: 0.4617
Epoch 8/10
439/439 [=====] - 132s 300ms/step - loss: 0.2409 - accuracy: 0.9047 - val_loss: 3.2305 - val_accuracy: 0.4570
Epoch 9/10
439/439 [=====] - 131s 298ms/step - loss: 0.2193 - accuracy: 0.9136 - val_loss: 3.1302 - val_accuracy: 0.4677
Epoch 10/10
439/439 [=====] - 136s 309ms/step - loss: 0.2141 - accuracy: 0.9201 - val_loss: 3.2673 - val_accuracy: 0.4817
```

Result: The second model shows testing accuracy of **48.17%**

Model 3: Added some more layers, activation function as tanh, 10 Epoch, training accuracy of 16.99%

```
# fitting the model
trained_model3 = model3.fit(df_train, validation_data = df_test, batch_size=50, validation_split=0.3, epochs =10)

Epoch 1/10
439/439 [=====] - 156s 342ms/step - loss: 1.8571 - accuracy: 0.1668 - val_loss: 1.7989 - val_accuracy: 0.1843
Epoch 2/10
439/439 [=====] - 147s 335ms/step - loss: 1.8170 - accuracy: 0.1607 - val_loss: 1.8055 - val_accuracy: 0.1670
Epoch 3/10
439/439 [=====] - 147s 335ms/step - loss: 1.8059 - accuracy: 0.1653 - val_loss: 1.8052 - val_accuracy: 0.1843
Epoch 4/10
439/439 [=====] - 149s 340ms/step - loss: 1.8018 - accuracy: 0.1677 - val_loss: 1.8021 - val_accuracy: 0.1843
Epoch 5/10
439/439 [=====] - 147s 334ms/step - loss: 1.8001 - accuracy: 0.1689 - val_loss: 1.7967 - val_accuracy: 0.1843
Epoch 6/10
439/439 [=====] - 147s 334ms/step - loss: 1.7976 - accuracy: 0.1724 - val_loss: 1.7993 - val_accuracy: 0.1843
Epoch 7/10
439/439 [=====] - 147s 334ms/step - loss: 1.7980 - accuracy: 0.1714 - val_loss: 1.7971 - val_accuracy: 0.1843
Epoch 8/10
439/439 [=====] - 147s 334ms/step - loss: 1.7973 - accuracy: 0.1699 - val_loss: 1.7971 - val_accuracy: 0.1670
Epoch 9/10
439/439 [=====] - 147s 334ms/step - loss: 1.7968 - accuracy: 0.1660 - val_loss: 1.7975 - val_accuracy: 0.1843
Epoch 10/10
439/439 [=====] - 147s 334ms/step - loss: 1.7968 - accuracy: 0.1699 - val_loss: 1.7957 - val_accuracy: 0.1843
```

Result: The third model shows testing accuracy of 18.43%

Model 4: Added some more layers, Activation function as leakyRelu, Adding Batch normalization, training accuracy of 95.21%.

```
# fitting the model
trained_model4 = model4.fit(df_train, validation_data = df_test, validation_split=0.3, batch_size=50, epochs =10, callbacks=early_stopping)

Epoch 1/10
439/439 [=====] - 190s 374ms/step - loss: 0.8634 - accuracy: 0.6769 - val_loss: 0.7505 - val_accuracy: 0.7337
Epoch 2/10
439/439 [=====] - 147s 334ms/step - loss: 0.5796 - accuracy: 0.7891 - val_loss: 0.6001 - val_accuracy: 0.7917
Epoch 3/10
439/439 [=====] - 147s 336ms/step - loss: 0.4658 - accuracy: 0.8323 - val_loss: 0.4939 - val_accuracy: 0.8273
Epoch 4/10
439/439 [=====] - 147s 335ms/step - loss: 0.3976 - accuracy: 0.8609 - val_loss: 0.5584 - val_accuracy: 0.8077
Epoch 5/10
439/439 [=====] - 147s 335ms/step - loss: 0.3327 - accuracy: 0.8833 - val_loss: 0.5400 - val_accuracy: 0.8217
Epoch 6/10
439/439 [=====] - 148s 336ms/step - loss: 0.2790 - accuracy: 0.9013 - val_loss: 0.6429 - val_accuracy: 0.7920
Epoch 7/10
439/439 [=====] - 147s 336ms/step - loss: 0.2333 - accuracy: 0.9185 - val_loss: 0.4705 - val_accuracy: 0.8473
Epoch 8/10
439/439 [=====] - 147s 335ms/step - loss: 0.1927 - accuracy: 0.9336 - val_loss: 0.5619 - val_accuracy: 0.8303
Epoch 9/10
439/439 [=====] - 147s 336ms/step - loss: 0.1630 - accuracy: 0.9440 - val_loss: 0.5252 - val_accuracy: 0.8443
Epoch 10/10
439/439 [=====] - 147s 336ms/step - loss: 0.1373 - accuracy: 0.9521 - val_loss: 0.5845 - val_accuracy: 0.8377
```

Accuracy:

```
model_accuracy(trained_model4)
```

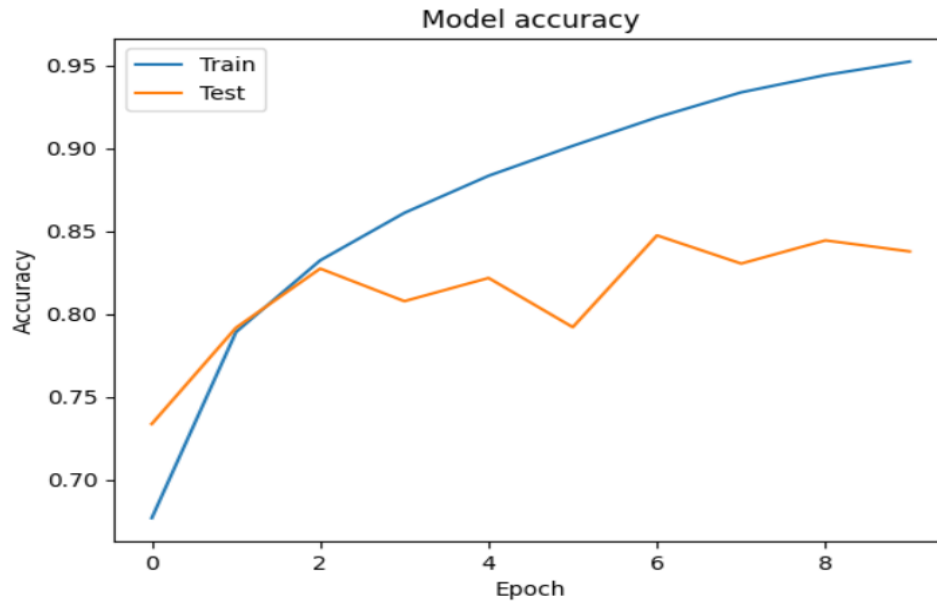


Fig: Intel Image classification dataset Accuracy Vs Epoch CNN Model

Loss:

```
model_loss(trained_model4)
```

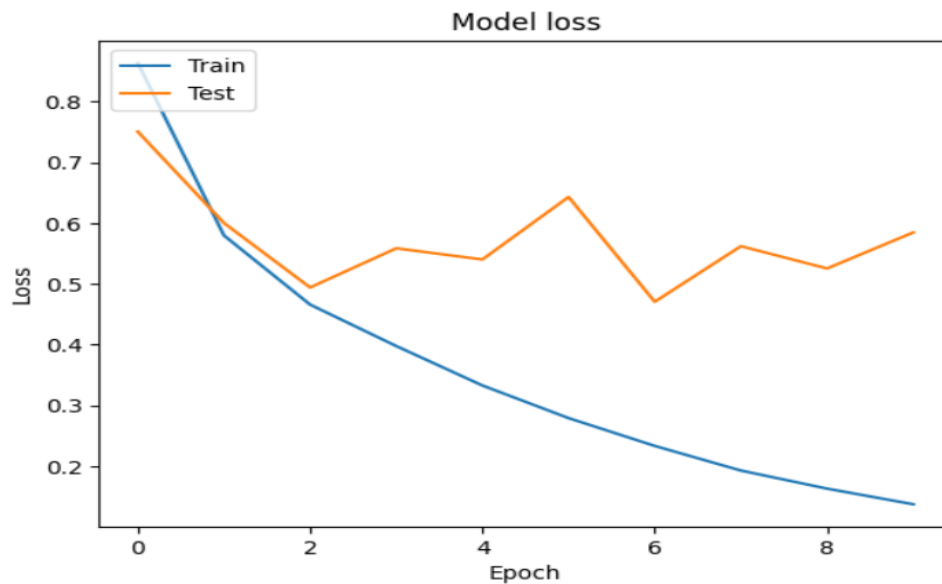


Fig: Intel Image classification dataset Model Loss Vs Epoch CNN Model

Result : The fourth model shows testing accuracy of **83.77%**

Model 5: Added some more layers, Activation function as Relu, Adding Batch Normalization, 10 Epoch, Training accuracy of 87.54%.

```
# fitting the model
trained_model5 = model5.fit(df_train, validation_data = df_test, validation_split=0.3, batch_size=50, epochs=10, callbacks = early_stopping)
```

Epoch 1/10
439/439 [=====] - 71s 136ms/step - loss: 1.1156 - accuracy: 0.5658 - val_loss: 4.1044 - val_accuracy: 0.2247
Epoch 2/10
439/439 [=====] - 53s 120ms/step - loss: 0.8101 - accuracy: 0.6992 - val_loss: 0.8774 - val_accuracy: 0.6900
Epoch 3/10
439/439 [=====] - 52s 118ms/step - loss: 0.6936 - accuracy: 0.7505 - val_loss: 1.1985 - val_accuracy: 0.5547
Epoch 4/10
439/439 [=====] - 53s 119ms/step - loss: 0.6004 - accuracy: 0.7847 - val_loss: 0.8524 - val_accuracy: 0.6757
Epoch 5/10
439/439 [=====] - 52s 118ms/step - loss: 0.5339 - accuracy: 0.8088 - val_loss: 0.5370 - val_accuracy: 0.8000
Epoch 6/10
439/439 [=====] - 52s 118ms/step - loss: 0.4917 - accuracy: 0.8236 - val_loss: 0.5821 - val_accuracy: 0.7883
Epoch 7/10
439/439 [=====] - 52s 118ms/step - loss: 0.4541 - accuracy: 0.8415 - val_loss: 0.5188 - val_accuracy: 0.8153
Epoch 8/10
439/439 [=====] - 52s 118ms/step - loss: 0.4125 - accuracy: 0.8529 - val_loss: 0.4599 - val_accuracy: 0.8370
Epoch 9/10
439/439 [=====] - 52s 118ms/step - loss: 0.3897 - accuracy: 0.8638 - val_loss: 0.7858 - val_accuracy: 0.7323
Epoch 10/10
439/439 [=====] - 52s 118ms/step - loss: 0.3473 - accuracy: 0.8754 - val_loss: 0.5201 - val_accuracy: 0.8193

Result : The fifth model shows testing accuracy of **81.93%**

Conclusion:

Convolutional Neural Networks (CNNs) have proven to be a powerful and effective tool for image classification tasks and the Intel Image Classification dataset serves as a valuable platform to demonstrate their capabilities. By offering efficient feature extraction, superior performance, scalability, transfer learning potential, and robustness to variations, CNNs provide significant advantages over traditional methods in this specific context.

Results: The best accuracy offered by basic CNN model on Intel Image Classification dataset is **83.77%**.

7.Transfer Learning Model:

Transfer learning is a powerful approach in deep learning that allows you to repurpose knowledge gained from a previously trained model (source model) on a new task (target task). It's akin to leveraging the expertise acquired in one subject to learn a related subject more efficiently. It is trained on dataset called as ImageNet which provides a wide variety of images classified under about 100 categories. The ImageNet dataset plays a crucial role in the development and success of transfer learning models. It is a massive collection of over 14 million labelled images categorized into thousands of different classes. This vastness and diversity expose the pre-trained models to a wide range of visual concepts, allowing them to learn generic features applicable to various tasks. The images in ImageNet are meticulously labelled with accurate and consistent annotations.

Working of transfer learning:

- **Train a source model:** You train a deep learning model, often on a large and general dataset like ImageNet, which contains millions of images and thousands of categories. This model learns to extract valuable features from images, such as edges, shapes, and textures.
- **Freeze some layers:** In the target task, you freeze the initial layers of the pre-trained model (usually convolutional layers that extract low-level features). These layers have already learned generic features applicable to various tasks and don't need to be retrained.
- **Fine-tune the final layers:** You add new layers or modify the final layers of the pre-trained model and train them on your specific dataset for the target task. These layers adapt the learned features to the specific categories you're interested in.

Benefits of transfer learning:

- **Faster training:** By leveraging pre-trained weights, you significantly reduce training time compared to training a model from scratch, especially for complex tasks with limited data.
- **Improved performance:** Transfer learning often leads to better performance on the target task, especially when dealing with small datasets where training from scratch might not be effective.
- **Reduced resource requirements:** Utilizing pre-trained models can be more efficient in terms of computational resources and memory, making it suitable for training on limited hardware.

Models trained on ImageNet:

1. VGG16:

Developed by the Visual Geometry Group (VGG) at Oxford University. A deep convolutional neural network with 16 convolutional layers, known for its good performance on image classification tasks.

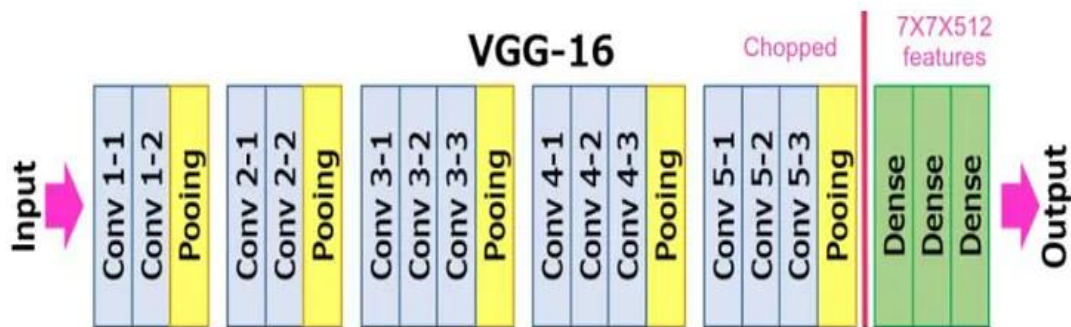


Fig: VGG 16 Architecture

Architecture:

Features a deep architecture with 16 convolutional layers, followed by pooling layers(5), fully connected layers, and a final softmax layer for classification.

Utilizes small filters (3x3) throughout the network, allowing for efficient learning of local features.

Strengths:

Good performance: VGG16 has achieved competitive accuracy on various image classification benchmarks, demonstrating its effectiveness in extracting relevant features from images.

Widely available: Pre-trained VGG16 models are readily accessible through popular deep learning libraries, making it convenient to apply transfer learning.

Relatively efficient: Compared to some deeper architectures, VGG16 requires less computational resources, making it suitable for training on moderate hardware.

Fit the model:

```
# fitting the model
trained_model_vgg = model_vgg.fit(train_generator3,
                                   validation_data=test_generator3,
                                   epochs=10,
                                   steps_per_epoch=len(train_generator3),
                                   validation_steps=len(test_generator3),
                                   callbacks=early_stopping
                                   )
```

```
Epoch 1/10
110/110 [=====] - 179s 2s/step - loss: 1.7021 - accuracy: 0.3141 - val_loss: 1.4910 - val_accuracy: 0.4493
Epoch 2/10
110/110 [=====] - 165s 1s/step - loss: 1.5268 - accuracy: 0.3846 - val_loss: 1.4600 - val_accuracy: 0.4750
Epoch 3/10
110/110 [=====] - 167s 2s/step - loss: 1.4899 - accuracy: 0.4033 - val_loss: 1.4249 - val_accuracy: 0.4810
Epoch 4/10
110/110 [=====] - 167s 2s/step - loss: 1.4759 - accuracy: 0.4137 - val_loss: 1.3876 - val_accuracy: 0.4963
Epoch 5/10
110/110 [=====] - 167s 2s/step - loss: 1.4641 - accuracy: 0.4204 - val_loss: 1.3692 - val_accuracy: 0.5137
Epoch 6/10
110/110 [=====] - 169s 2s/step - loss: 1.4538 - accuracy: 0.4217 - val_loss: 1.3731 - val_accuracy: 0.4993
Epoch 7/10
110/110 [=====] - 168s 2s/step - loss: 1.4433 - accuracy: 0.4303 - val_loss: 1.3510 - val_accuracy: 0.5063
Epoch 8/10
110/110 [=====] - 170s 2s/step - loss: 1.4342 - accuracy: 0.4344 - val_loss: 1.3595 - val_accuracy: 0.5163
Epoch 9/10
110/110 [=====] - 169s 2s/step - loss: 1.4288 - accuracy: 0.4437 - val_loss: 1.3396 - val_accuracy: 0.5217
Epoch 10/10
110/110 [=====] - 169s 2s/step - loss: 1.4193 - accuracy: 0.4451 - val_loss: 1.3278 - val_accuracy: 0.5273
```

Fig: VGG 16 Model fit on Intel Image Classification dataset

Accuracy:

```
model_accuracy(trained_model_vgg)
```

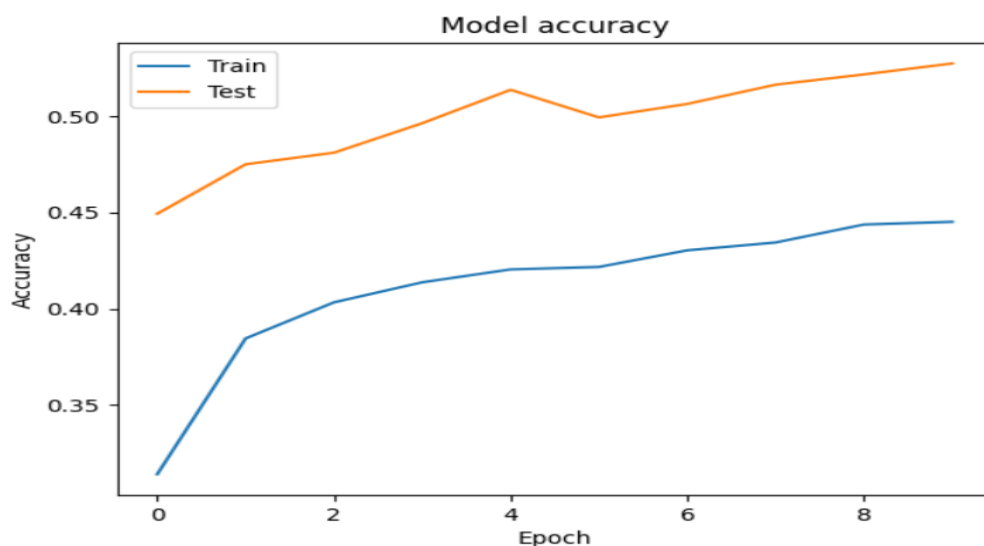


Fig: Intel Image classification dataset Accuracy Vs Epoch for VGG 16

Loss:

```
model_loss(trained_model_vgg)
```

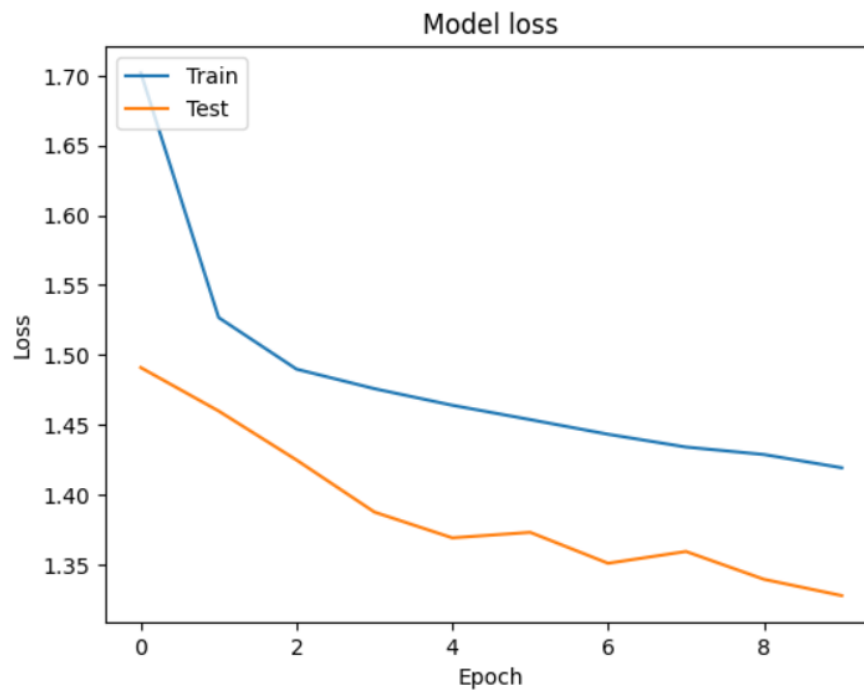


Fig: Intel Image classification dataset Model Loss Vs Epoch for VGG 16

Result: The accuracy of VGG 16 on Intel Image classification dataset is 52.73%.

2. Mobile Net V2:

MobileNet V2 is a lightweight convolutional neural network (CNN) architecture well-suited for transfer learning applied to the Intel Image Classification dataset.



The characteristics are:

- Designed for mobile and embedded devices: Employs depth-wise separable convolutions and linear bottleneck layers to achieve high accuracy with reduced computational cost and memory footprint.
- Pre-trained on ImageNet: Like other transfer learning models, Mobile Net V2 leverages pre-trained knowledge from the vast ImageNet dataset, learning fundamental visual features.

Transfer Learning for Intel Image Classification:

- Freeze initial layers: The pre-trained weights of the initial layers in MobileNet V2 are frozen, preserving their generic feature extraction capabilities.
- Fine-tune final layers: The final layers of the network are either modified or replaced with new layers specific to the Intel dataset categories (buildings, forests, glaciers, etc.).
- Training on Intel dataset: The modified MobileNet V2 is then trained on the Intel dataset, allowing the final layers to adapt the learned features for classifying these specific image categories.

Benefits of using MobileNet V2:

- Efficiency: Compared to heavier models like VGG16, MobileNet V2 requires less computational resources and memory, making it suitable for training on devices with limited capabilities.
- Accuracy: Despite its lightweight nature, MobileNet V2 has demonstrated competitive accuracy on various image classification benchmarks.

- **Transferability:** The pre-trained knowledge from ImageNet provides a solid foundation for learning relevant features, potentially improving classification performance on the Intel dataset.

Fit The model:

```
trained_model_mobilenet = model_mob.fit(train_generator2,
                                         validation_data=test_generator2,
                                         epochs=10,
                                         steps_per_epoch=len(train_generator2),
                                         validation_steps=len(test_generator2),
                                         callbacks=callbacks
                                         )
```

Epoch 1/10
 110/110 [=====] - ETA: 0s - loss: 0.6121 - accuracy: 0.7718/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103
 saving_api.save_model(
 110/110 [=====] - 105s 906ms/step - loss: 0.6121 - accuracy: 0.7718 - val_loss: 0.3421 - val_accuracy: 0.8720 - lr: 0.0010
 Epoch 2/10
 110/110 [=====] - 89s 810ms/step - loss: 0.4068 - accuracy: 0.8504 - val_loss: 0.3111 - val_accuracy: 0.8863 - lr: 0.0010
 Epoch 3/10
 110/110 [=====] - 87s 792ms/step - loss: 0.3674 - accuracy: 0.8677 - val_loss: 0.3219 - val_accuracy: 0.8843 - lr: 0.0010
 Epoch 4/10
 110/110 [=====] - 88s 804ms/step - loss: 0.3541 - accuracy: 0.8735 - val_loss: 0.3144 - val_accuracy: 0.8777 - lr: 0.0010
 Epoch 5/10
 110/110 [=====] - 88s 797ms/step - loss: 0.3488 - accuracy: 0.8710 - val_loss: 0.2945 - val_accuracy: 0.8937 - lr: 0.0010
 Epoch 6/10
 110/110 [=====] - 87s 793ms/step - loss: 0.3395 - accuracy: 0.8738 - val_loss: 0.2782 - val_accuracy: 0.8970 - lr: 0.0010
 Epoch 7/10
 110/110 [=====] - 86s 780ms/step - loss: 0.3264 - accuracy: 0.8805 - val_loss: 0.2984 - val_accuracy: 0.8857 - lr: 0.0010
 Epoch 8/10
 110/110 [=====] - 87s 791ms/step - loss: 0.3245 - accuracy: 0.8812 - val_loss: 0.2806 - val_accuracy: 0.8953 - lr: 0.0010
 Epoch 9/10
 110/110 [=====] - 87s 795ms/step - loss: 0.3162 - accuracy: 0.8853 - val_loss: 0.2871 - val_accuracy: 0.8957 - lr: 0.0010
 Epoch 10/10
 110/110 [=====] - 87s 789ms/step - loss: 0.3106 - accuracy: 0.8847 - val_loss: 0.2785 - val_accuracy: 0.8970 - lr: 0.0010

Fig: MobileNet -V2 Model fit on Intel Image Classification dataset

Accuracy:

```
model_accuracy(trained_model_mobilenet)
```

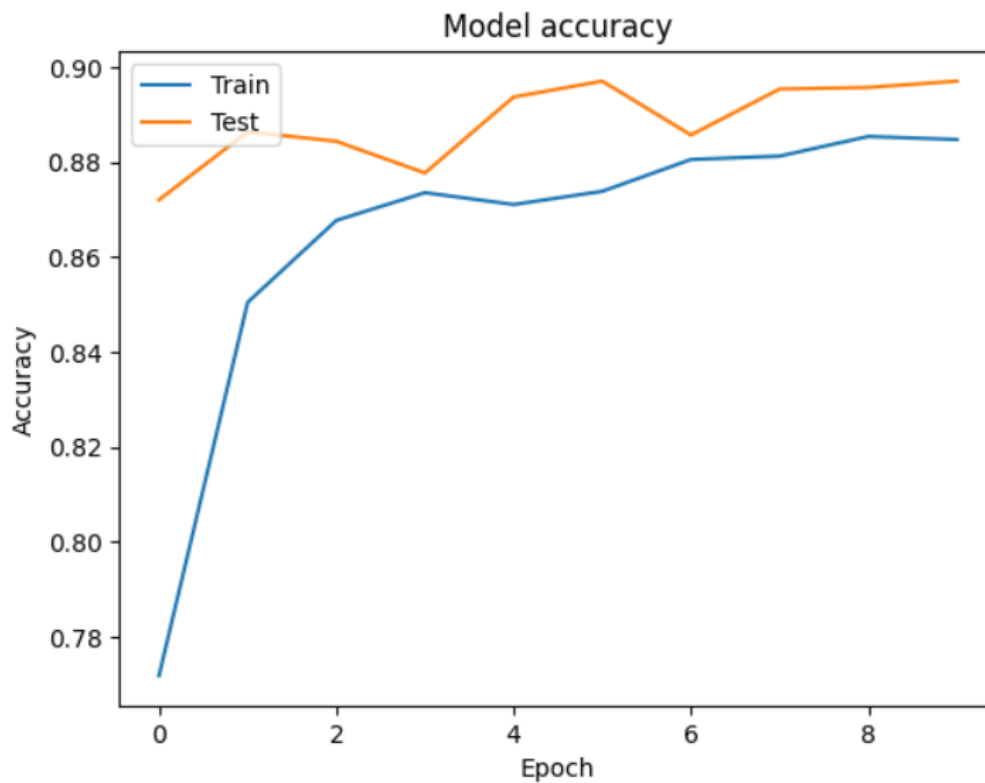


Fig: Intel Image classification dataset Accuracy Vs Epoch for MobileNet V2

Loss:

```
model_loss(trained_model_mobilenet)
```

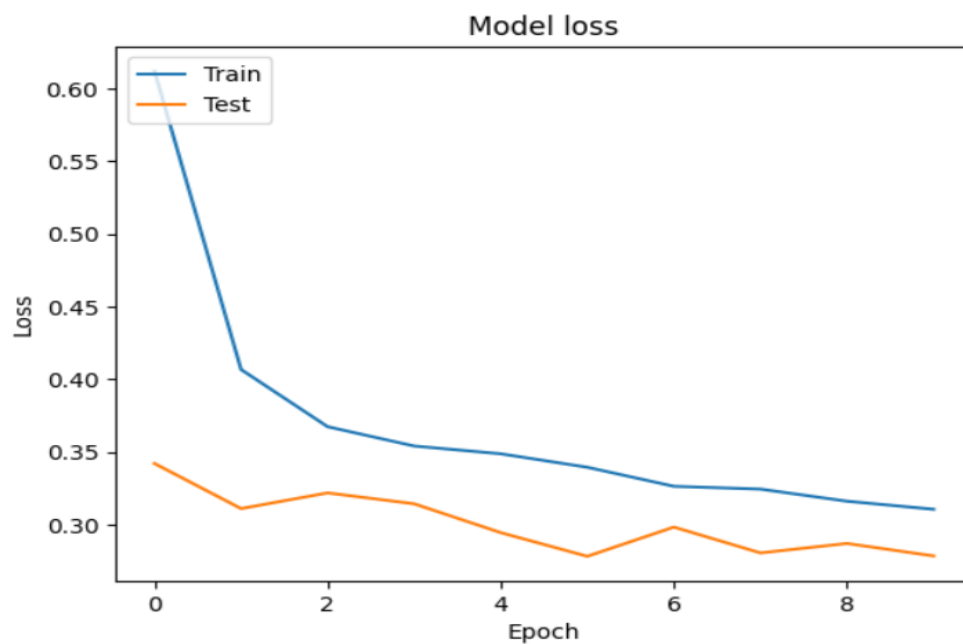
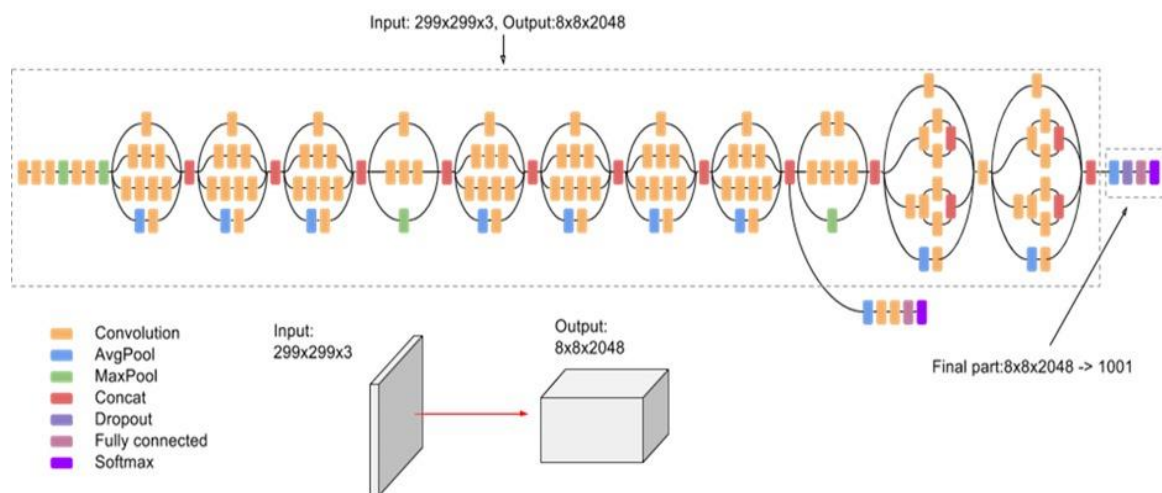


Fig: Intel Image classification dataset Model Loss Vs Epoch for MobileNet V2

Result: The accuracy on Intel Image classification is 89.70% .

3. Inception V3:

Inception V3 is a powerful deep learning architecture frequently employed for transfer learning in image classification tasks.



Architecture:

- Developed by Google researchers, Inception V3 utilizes a unique architecture with inception modules that combine convolutional and pooling layers within a single unit.
- These modules allow for efficient feature extraction at different scales and resolutions, potentially capturing richer information from images.
- The network also incorporates techniques like batch normalization to improve training stability and reduce the need for complex hyperparameter tuning.

Strengths:

- High accuracy: Inception V3 has demonstrated state-of-the-art performance on various image classification benchmarks, suggesting its ability to learn discriminative features effectively.
- Efficient use of parameters: Despite its depth, Inception V3 achieves good performance with a relatively smaller number of parameters compared to some other architectures, making it potentially more efficient for training.
- Readily available: Pre-trained Inception V3 models are widely accessible through popular deep learning libraries, facilitating its use in transfer learning scenarios.

Fit the model:

```
trained_model_inception = model_incept.fit(train_generator1,
                                           validation_data=test_generator1,
                                           epochs=10,
                                           steps_per_epoch=len(train_generator1),
                                           validation_steps=len(test_generator1),
                                           callbacks=callbacks
                                           )
```

```
Epoch 1/10
110/110 [=====] - 365s 3s/step - loss: 3.5234 - accuracy: 0.7921 - val_loss: 0.7963 - val_accuracy: 0.8940 - lr: 0.0010
Epoch 2/10
110/110 [=====] - 306s 3s/step - loss: 1.1633 - accuracy: 0.8662 - val_loss: 1.0102 - val_accuracy: 0.8753 - lr: 0.0010
Epoch 3/10
110/110 [=====] - 305s 3s/step - loss: 1.2976 - accuracy: 0.8676 - val_loss: 1.0419 - val_accuracy: 0.8977 - lr: 0.0010
Epoch 4/10
110/110 [=====] - ETA: 0s - loss: 1.1721 - accuracy: 0.8840
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.000100000000474974513.
110/110 [=====] - 308s 3s/step - loss: 1.1721 - accuracy: 0.8840 - val_loss: 1.1508 - val_accuracy: 0.8837 - lr: 0.0010
Epoch 5/10
110/110 [=====] - ETA: 0s - loss: 0.8058 - accuracy: 0.9091Restoring model weights from the end of the best epoch: 1.
110/110 [=====] - 315s 3s/step - loss: 0.8058 - accuracy: 0.9091 - val_loss: 0.8244 - val_accuracy: 0.9097 - lr: 1.0000e-04
Epoch 5: early stopping
```

Fig: MobileNet -V2 Model fit on Intel Image Classification dataset

Accuracy:

```
model_accuracy(trained_model_inception)
```

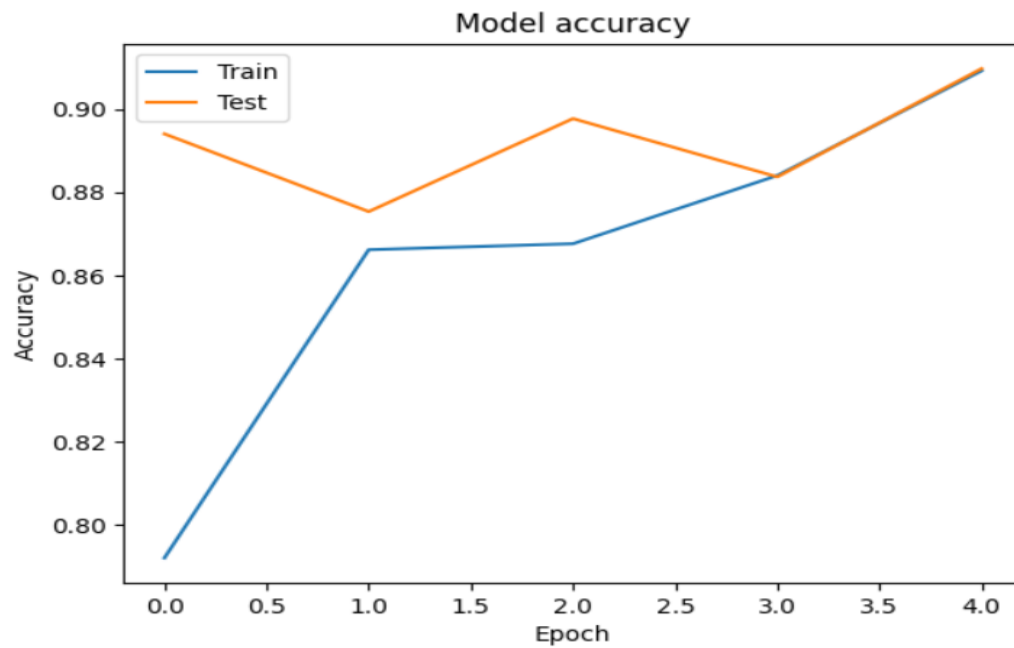


Fig: Intel Image classification dataset Accuracy Vs Epoch for Inception V3

Loss:

```
model_loss(trained_model_inception)
```

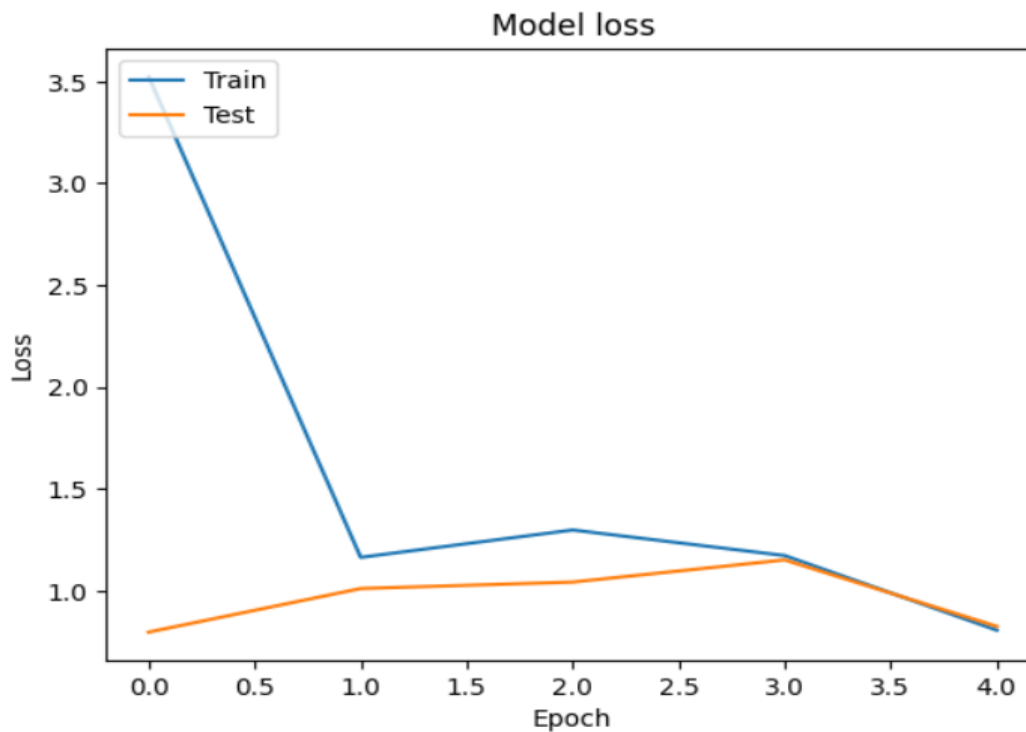


Fig: Intel Image classification dataset Accuracy Vs Epoch for Inception V3

Result: The accuracy of Inception v3 on Intel Image Classification Dataset is 90.97%.

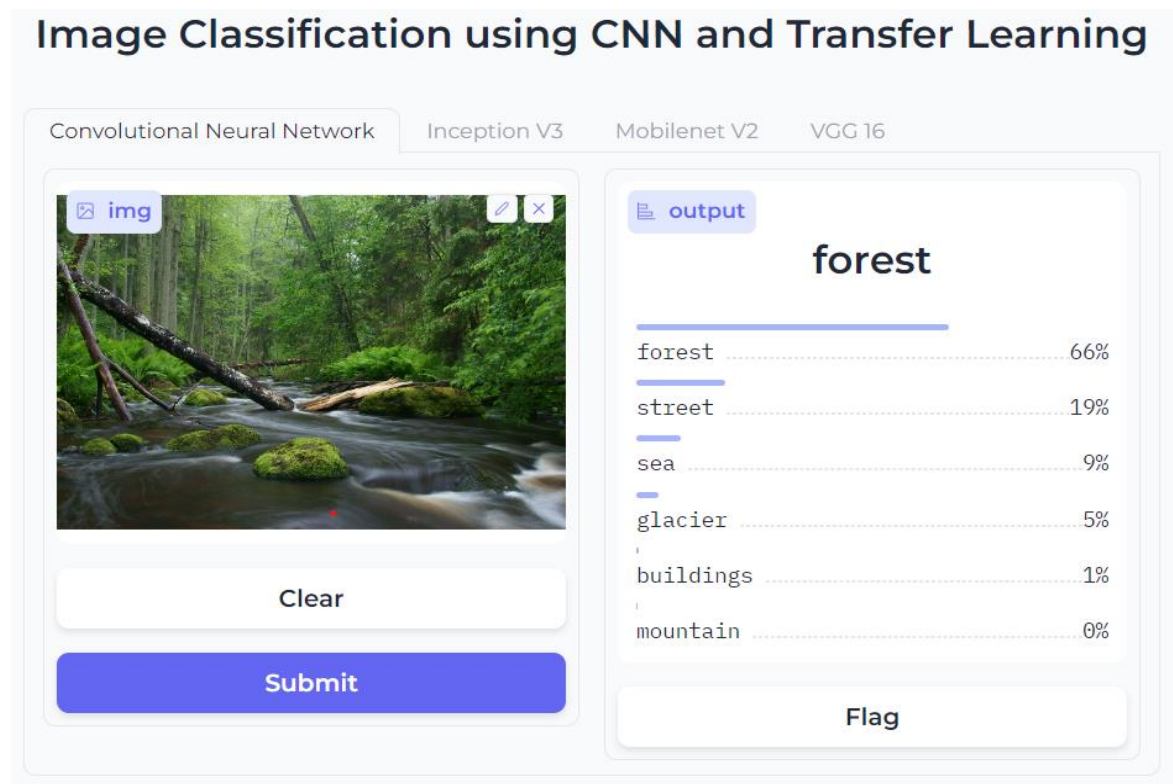
8. Gradio:

With the help of the Python package Gradio, we can design user interfaces for machine learning models. It's frequently used to build interactive applications or demos that let users to enter data and view real-time predictions from the model. Using Gradio with the Intel Image Classification dataset is done as follows:

- Load the Intel Image Classification Dataset: This dataset typically consists of images classified into several categories like buildings, forests, mountains, etc.
- Train a Model: For image classification tasks, convolutional neural networks (CNNs) and Transfer Learning is used.
- Create a Gradio Interface: After training the model, a simple interface can be created using Gradio such that any image could be uploaded by the user to check the model's prediction. A function can be defined that takes an image, runs through the models as we defined and returns predictions as output.

The outcomes of all models used in the projects are as follows:

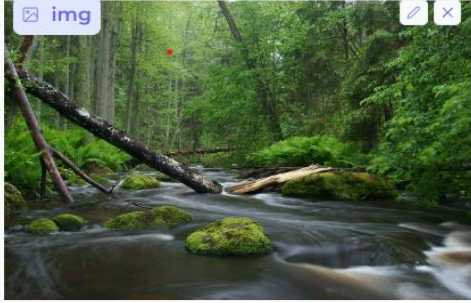
1. CNN best fit model:



2. Inception V3:

Image Classification using CNN and Transfer Learning

Convolutional Neural Network Inception V3 Mobilenet V2 VGG 16



img

Clear

Submit

output

forest


forest	100%
mountain	0%
buildings	0%
glacier	0%
sea	0%
street	0%

Flag

3. Mobile Net V2:

Image Classification using CNN and Transfer Learning

Convolutional Neural Network Inception V3 Mobilenet V2 VGG 16



img

Clear

Submit

output

sea


sea	99%
mountain	0%
glacier	0%
buildings	0%
forest	0%
street	0%

Flag

4. VGG16:

Image Classification using CNN and Transfer Learning

Convolutional Neural Network Inception V3 Mobilenet V2 **VGG 16**



img

Clear

Submit

output

sea

sea	98%
forest	2%
street	0%
buildings	0%
glacier	0%
mountain	0%

Flag

9. CONCLUSION AND FUTURE SCOPE:

Conclusion: Comparative Analysis of CNNs, Transfer Learning, and Inception V3 for Intel Image Classification

This analysis compared the performance of Convolutional Neural Networks (CNNs), transfer learning models, and specifically Inception V3 in the context of the Intel Image Classification dataset.

Overall, transfer learning, particularly with Inception V3, emerged as the most effective approach for image classification on the Intel dataset. It offers significant advantages in terms of:

- **Faster training:** Leveraging pre-trained knowledge significantly reduces training time compared to training a CNN from scratch.
- **Improved accuracy:** Transfer learning often leads to better classification performance, especially on limited datasets, by utilizing pre-trained features relevant to the target task.
- **Reduced resource requirements:** By fine-tuning only a portion of the model, transfer learning requires less computational resources compared to training a full CNN from scratch.

It's crucial to recognize that: Selecting the best pre-trained model relies on the particular task and the features of the dataset. Comparing other models such as ResNets or VGG 16 may be worthwhile. The model architecture and hyperparameters must be fine-tuned in order to achieve optimal transfer learning performance.

In conclusion, even though CNNs are still powerful tools for image classification, transfer learning—especially with Inception V3—offers a more effective and efficient method for tasks like the Intel Image Classification dataset because of its faster training, better accuracy, and lower resource requirements.

In this project we have successfully implemented CNN and Transfer Learning Model on our dataset and their comparative analysis. The outcomes suggest that with an accuracy of 90.37%, Inception V3 from transfer learning model predicts the most accurate classification of Intel Image Classification Dataset.

Inferential Outcomes:

Model	Training Accuracy	Testing Accuracy
CNN	0.9397	0.8433
InceptionV3	0.9129	0.9037
MobileNetV2	0.8846	0.8923
VGG16	0.8462	0.8627

Future Scope:

1. Examining Complex Transfer Learning Methods:
 - Fine-tuning strategies: Performance may be enhanced beyond the fundamental fine-tuning used in this analysis by experimenting with various approaches, such as freezing particular layers, adjusting learning rates for distinct layers, or applying knowledge distillation techniques.
 - Meta-learning: For jobs with little data, such as the Intel dataset, using meta-learning algorithms that can learn how to learn from few-shot datasets may be helpful.
 - Domain-specific pre-training: Additional performance gains may be obtained by investigating pre-trained models created especially for scene classification tasks or by bridging the gap between ImageNet and the Intel dataset with methods like domain adaption.
2. Investigating Data Augmentation Techniques:
 - Data augmentation strategies: Applying various data augmentation techniques like random cropping, flipping, rotations, and colour jittering can artificially increase the size and diversity of the training data, potentially improving the generalization ability of both CNNs and transfer learning models.
 - Semi-supervised learning: Utilizing unlabelled data from the Intel dataset alongside the labelled data could be explored through semi-supervised learning techniques to further enhance model performance.
3. Exploring Alternative Architectures and Techniques:
 - Newer pre-trained models: Investigating the effectiveness of more recent and advanced pre-trained models like Efficient Net or Swin Transformer on the Intel dataset could potentially lead to superior performance compared to VGG16.

- Ensemble methods: Combining predictions from multiple CNN or transfer learning models using ensemble methods like bagging or boosting could potentially improve robustness and accuracy.
4. Addressing Ethical Considerations:
- Bias in pre-trained models: Exploring and mitigating potential biases present in pre-trained models like VGG16, which could impact the fairness and generalizability of the results, is crucial for responsible development and deployment of these models.
 - Explainability and interpretability: Developing techniques to understand how CNNs and transfer learning models arrive at their predictions, especially in the context of the Intel dataset, can be valuable for improving trust and transparency in these models.

By actively exploring these future directions, researchers and developers can continue to push the boundaries of image classification using CNNs, transfer learning, and Inception V3, leading to more accurate, efficient, and reliable solutions for various real-world applications.

10.References

- [1] Understanding of a convolutional neural network, 10.1109/ICEngTechnol.2017.8308186, Saad Albawi, Tareq Abed Mohammed, Saad Al-Zawi,
[<https://ieeexplore.ieee.org/document/8308186>]
- [2] Image Captioning Using Inception V3 Transfer Learning Model,
10.1109/ICCES51350.2021.9489111, Sheshang Degadwala, Dhairya Vyas, Haimanti Biswas, Utsho Chakraborty, Sowrav Saha,
[<https://ieeexplore.ieee.org/document/9489111>]
- [3] TTransfer learning using VGG-16 with Deep Convolutional Neural Network for Classifying Images, 10.29322/IJSRP.9.10.2019.p9420, Srikanth Tammina,
[https://www.researchgate.net/publication/337105858_Transfer_learning_using_VGG-16_with_Deep_Convolutional_Neural_Network_for_Classifying_Images]
- [4] Kaggle
[<https://www.kaggle.com/datasets/puneet6060/intel-image-classification/code>]