Table of Contents:

ABSTRACT

The goal of this study is to find how transfer learning might enhance a deep learning model's accuracy in categorizing dog breeds. In this study, a transfer learning method employing MobileNetV2, a pre-trained model that was first trained on the ImageNet dataset, is compared against a CNN model created from scratch by us. The goal of the study is to assess the trade-offs between creating a model from scratch and applying transfer learning by contrasting the effectiveness of these two methodologies. The popularity of transfer learning in the field of deep learning, as well as the difficulties and opportunities given by the task of identifying dog breeds, are the driving forces behind the choice of this topic. Dog breed categorization is a frequent and difficult subject in computer vision since there are several breeds that can resemble one another rather closely. Therefore, even for humans, let alone a machine learning model, it might be challenging to correctly identify the breed of a dog in an image. The initiative also aims to further scientific knowledge of transfer learning and the trade-offs between creating a model from scratch and employing one that has already been trained. This study compares the performance of a CNN model created from scratch with transfer learning using MobileNetV2, as well as the correctness of the models as a result of various data augmentation methods. By performing different operations on the original photos, such as rotating, flipping, and zooming, data augmentation is a method for artificially expanding the training dataset. The study also intends to investigate the effects of several hyper-parameters on the effectiveness of the models, including learning rate and batch size. The study seeks to offer insights into the best approaches for creating efficient deep learning models for image classification tasks by assessing these variables. The project aims to learn and prove several things. First, it seeks to evaluate the effectiveness of transfer learning in improving the accuracy of a deep learning model for the task of classifying dog breeds. Second, it aims to evaluate the trade-offs involved in building a model from scratch versus using transfer learning, such as the time and computational resources required, the amount of data needed to train a model effectively, and the impact of hyper-parameters on model performance. Third, the project seeks to contribute to the scientific understanding of transfer learning and deep learning for image classification, with the goal of developing more effective and efficient techniques for solving similar problems in the future. In conclusion, the goal of this study is to investigate and contrast several methods for developing deep learning models for the classification of dog breeds. The goal of the research is to further our understanding of deep learning for image classification by assessing the efficiency of transfer learning and the trade-offs involved in creating a model from scratch as opposed to utilizing one that has already been trained. The project's ultimate goal is to provide improved methods for resolving issues of this nature in the future.

INTRODUCTION

In many fields, such as animal welfare, veterinary medicine, and animal research, classifying dog breeds is an essential task. Identifying a dog's breed based on a photograph may provide crucial information about its behavior, health, and genetic makeup. Machine learning techniques have been employed for this purpose more and more in recent years, with transfer learning and convolutional neural networks (CNNs) among the most often used approaches.

Transfer learning is utilizing a model that has already been trained on a huge dataset and refining it on a smaller dataset for a particular task. On the other hand, CNNs are deep learning models that are intended to automatically analyse pictures and extract information. The primary objective of the study is to investigate and contrast several methods for applying deep learning to categorize dog breeds. Due to the enormous variety of dog breeds and the fact that many varieties share physical characteristics, classifying dog breeds can be difficult. Therefore, classifying dog breeds using machine learning algorithms might be a useful tool for those working in the pet market.

Regression and classification are the further divisions of supervised learning techniques. In order to forecast a continuous output variable, such as a house's price based on its features, regression models are utilized. On the other hand, classification models are used to forecast a discrete output variable, such the breed of a dog based on a picture. Both the transfer learning strategy employing MobileNetV2 and the CNN model created from scratch use supervised learning algorithms, although their learning methodologies differ. The CNN model created from scratch employs a more complicated architecture to learn the characteristics of the input pictures and needs a bigger volume of labeled data to train.

In this project, scaling, normalization, and augmentation will all be used to preprocess dog image data from a publicly available dataset. Two models will then be trained using the dataset, and the effectiveness of each model will be evaluated using a range of metrics including accuracy, precision, recall, and F1 score. Finally, we'll look at how the CNN and transfer learning models performed and discuss their benefits and drawbacks.

The findings of this study may be useful in the fields of animal welfare, veterinary medicine, and animal research. They will add to the body of information on the classification of dog breeds using machine learning techniques.This study has the potential to help achieve these objectives by creating two machine learning models for classifying dog breeds, contrasting them, and highlighting the advantages and disadvantages of each method.

The use of artificial intelligence (AI) tools, which enable the development and evaluation of machine learning models for dog breed classification, is essential to the project's success. AI technologies provide a range of capabilities for handling photo data, including preprocessing, feature extraction, model training, and evaluation.Another important AI method applied in this study is convolutional neural networks (CNNs), which are deep learning models built to automatically process images and extract features. CNNs are especially well-suited for image classification tasks like identifying different dog breeds because they can learn to recognize patterns and characteristics in images that are crucial for making this distinction.

Other AI technologies employed in this research in addition to transfer learning and CNNs are data pretreatment methods including scaling, normalization, and augmentation, as well as assessment metrics like accuracy, precision, recall, and F1 score. The construction, training, and evaluation of machine learning models for classifying dog breeds depend heavily on each of these technologies.

DATA DESCRIPTION

1. Stanford Dogs dataset

For the Stanford Dogs dataset, the Stanford University Vision and Learning Lab gathered a huge collection of annotated dog pictures. The collection contains close to 20,000 images, with each breed of dog having between 150 and 200 pictures. Each image has accurate and reliable labels because each photo was individually tagged by a person.

The images in the collection are diverse in terms of their sizes, formats, and resolutions, and some of them include numerous dogs or dogs in elaborate settings. Before using the dataset for machine learning, it is usual to preprocess the images to a standard size , normalize the pixel values, and augment the data to increase its variability.

The Stanford Dogs dataset, which has been widely used in studies on dog breed categorization, is one of the benchmark datasets for this project. It is available on the Stanford University website for free download.

Format: JPG

Size: The images vary in size, with some as small as 100 x 100 pixels and others as large as 500 x 500 pixels.

Quality: The camera used to take the photos and other factors can affect the image quality. Low contrast, noise, and other imperfections in some photos might degrade their quality.

Sampling: The sample is not uniform or random because the images in the dataset were gathered from a variety of places, including internet image repositories and private collections. It's possible that some dog breeds are underrepresented in the collection or that some breeds have more photos than others.

Noise: Particularly those taken in dimly lit or highly contrasted environments may have noise or other artifacts in their images. The dataset doesn't offer any details regarding noise levels or other aspects of the images.

Dataset weblink: https://www.tensorflow.org/datasets/catalog/stanford_dogs

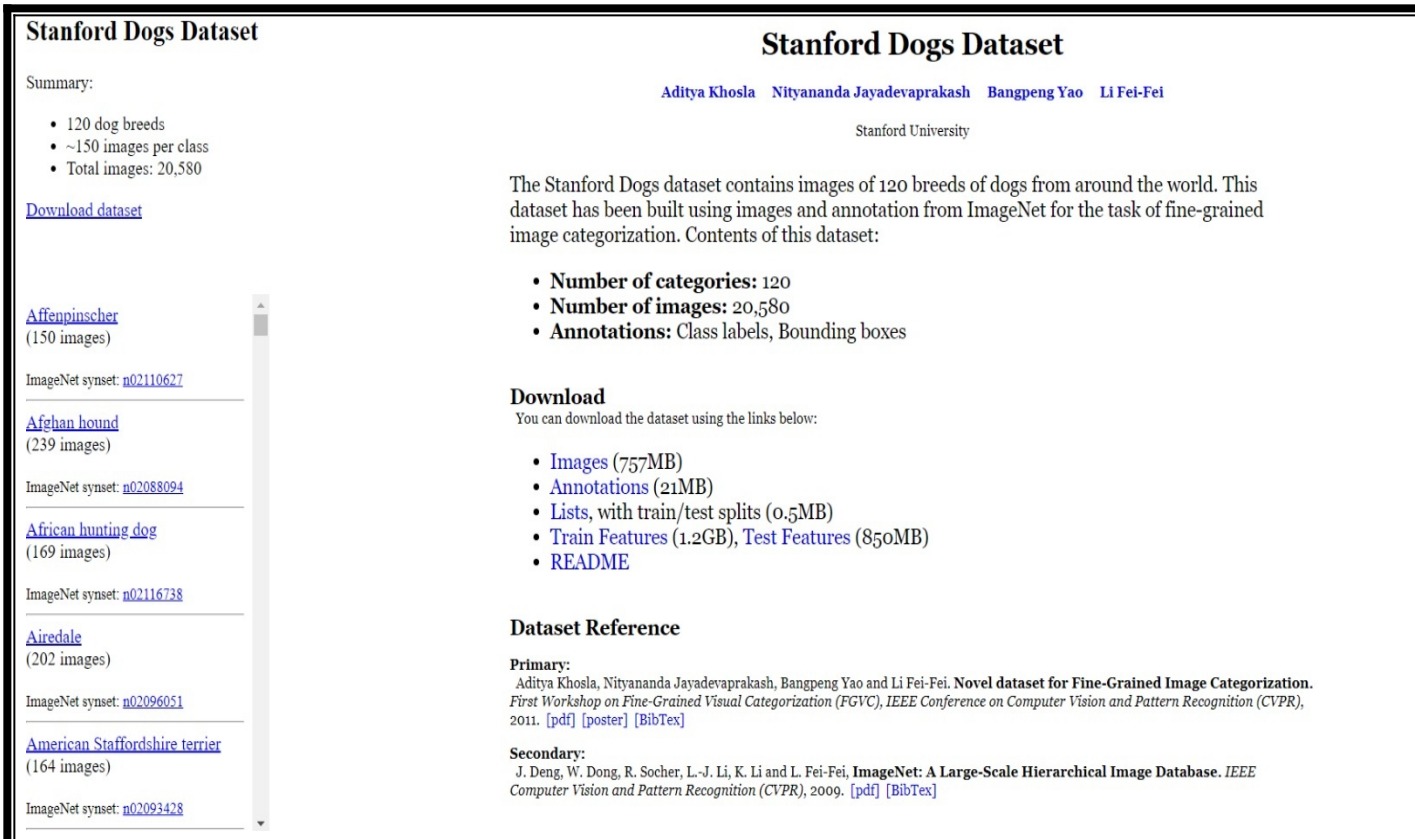


Image 1: Snapshot of the Stanford Dogs Dataset

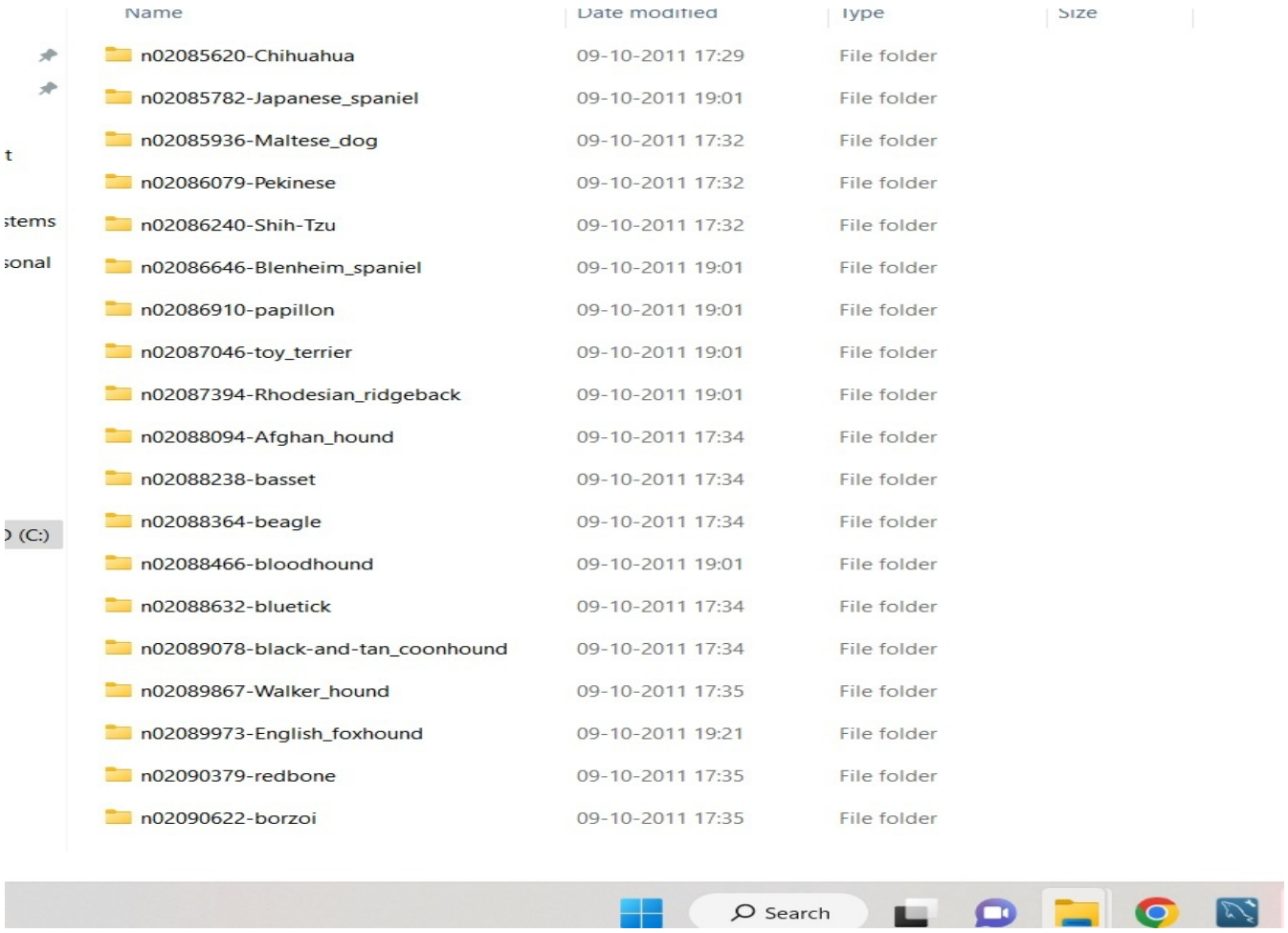


Image 2: Downloaded Dataset in PC

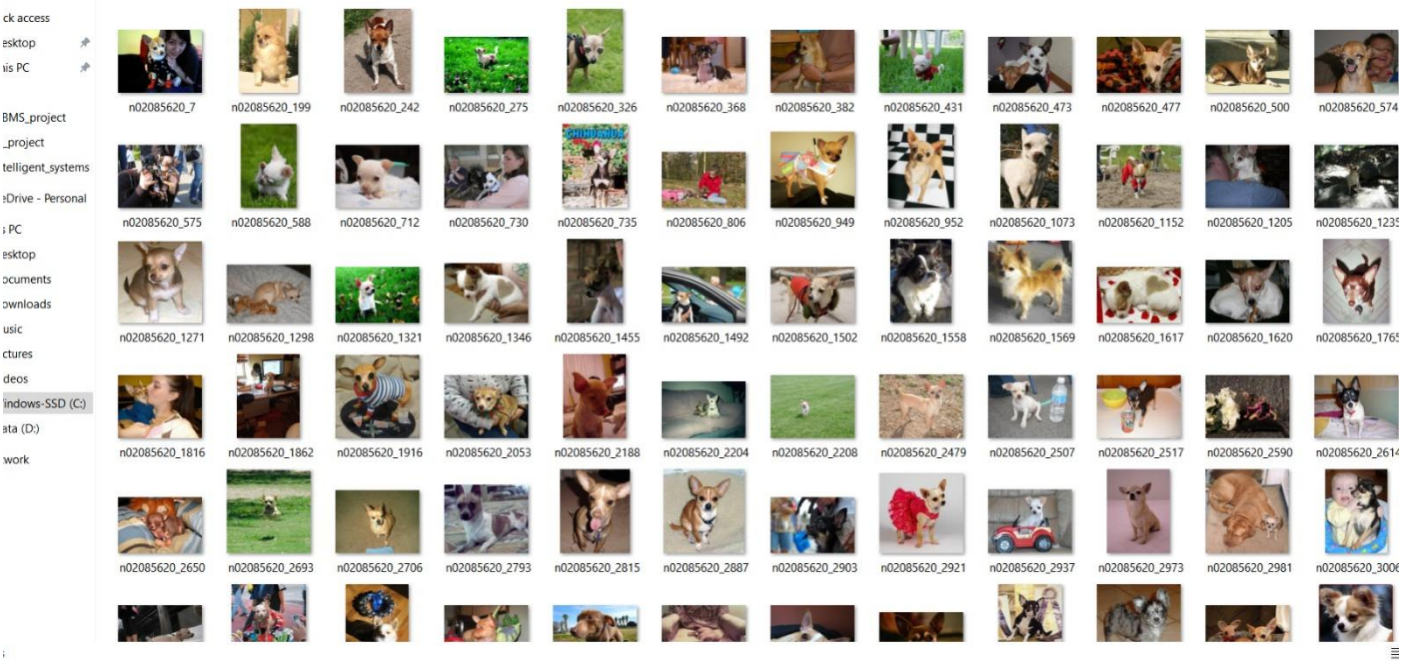


Image 3: Dogs Images from the Dataset

2. 1,000 classes dataset used for MobileNetV2

For mobile and embedded devices, a specific neural network architecture known as MobileNetV2 was developed. For effective inference on these devices, MobileNetV2 is frequently trained on fewer datasets than other deep learning models. One such dataset is the 1,000 classes dataset, which contains 1,000 different object categories, such as animals, cars, and household goods.

The images in the 1,000 classes dataset are comparable to those in the ImageNet collection despite having fewer categories. About 1,000 images in each category have undergone preprocessing and been given a standard size. MobileNetV2 has been trained using the dataset to perform a number of tasks, such as object detection, image segmentation, and classification. On mobile devices, it is often used for neural network training and evaluation.

Although the 1,000 classes dataset was not created explicitly for the objective of classifying dog breeds, it can be used to train and assess models for it, especially when used in conjunction with transfer learning strategies.

Format: JPG
Size: The images are all re-sized to a standard size of 224 x 224 pixels.
Quality: The collection contains high-quality images with little noise or artifacting. Images have undergone preprocessing to get rid of any distortions or compression artifacts that can degrade their quality.
Sampling: Animals, cars, and household goods are just a few of the object types represented in the dataset's sampled images. There are around 1,000 images in each category, striking a fair balance between quantity and diversity.
Noise: To eliminate any noise or other artifacts that can degrade the quality of the images in the dataset, preprocessing was applied to the images. Even yet, some images might still have noise or other flaws, especially if they were taken in dimly lit or highly contrasted environments.

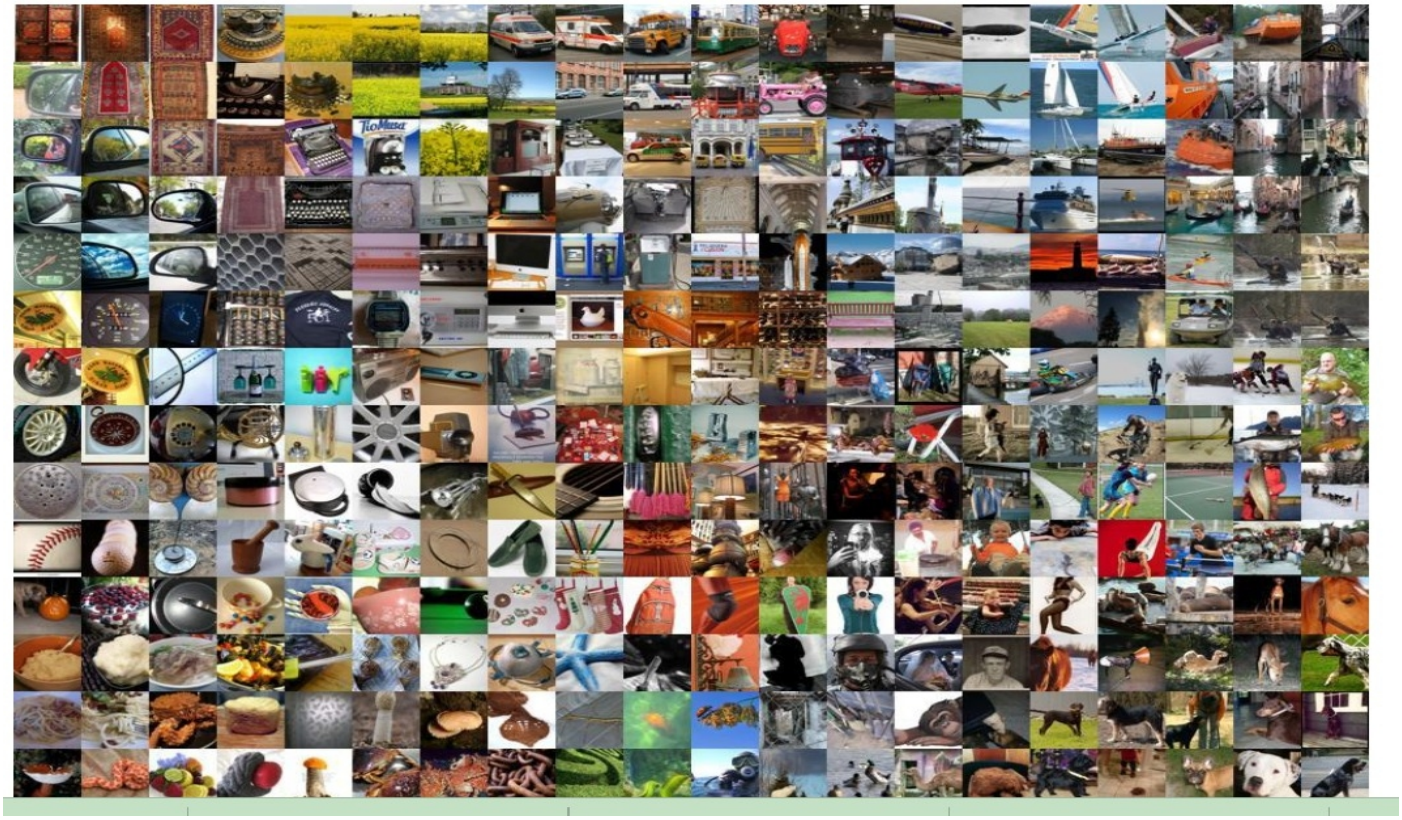


Image 4: Images from Imagenet Dataset

ALGORITHMS USED

1. CONVOLUTIONAL NEURAL NETWORK

An advanced form of neural network architecture known as a CNN is designed to operate with data that has a grid-like pattern, such as images. Through the use of several layers of convolutional and pooling techniques, the local features in the input image are retrieved. Subsequent layers combine these local features with higher-level features to represent the input image hierarchically.

As CNNs are made to work with input that has a grid-like layout, like photographs, they are especially well suited for image classification jobs. When classifying dog breeds, dog photos are the input data points, and the labels relate to the breed of dog in the image.
The project's CNN architecture is made up of several layers of convolutional and pooling algorithms. Local features are extracted from the input picture by applying a separate learnable filter to each convolutional layer in the architecture. The filters become adept at identifying particular patterns or characteristics, such as edges, corners, and forms. The CNN can learn increasingly complicated and abstract characteristics, such as the forms of various body parts, by layering many convolutional layers.

A rectified linear unit (ReLU) activation function is applied to the output following each convolutional layer. ReLU gives the model nonlinearity, which enables it to learn intricate connections between the input data and output labels. Convolutional layer output is then flattened into a vector and sent into a fully linked layer. The extracted features are moved from the input space of class probabilities to the fully connected layer's output space using a softmax activation function. A vector of probabilities representing the chance that the input image belongs to each of the potential dog breeds is the model's output.

In order to improve the model during training, a loss function that gauges the discrepancy between expected output probabilities and the actual labels is minimized. In order to reduce loss and increase the model's accuracy on the training set of data, the optimization method modifies the model's parameters, including the filter weights.

After the model has been trained, additional dog photos may be classified by feeding them into the model to get the anticipated class probabilities. The estimated breed of the dog in the photograph is then chosen based on the highest likelihood class.

2. TRANSFER LEARNING

The pre-trained MobileNetV2 architecture is used as a feature extractor for the job of categorizing dog breeds in the transfer learning technique. The pre-trained network can distinguish fundamental forms, edges, and textures, which are helpful in differentiating dog breeds. The pre-trained network has previously learnt a wide range of characteristics from a large-scale dataset. The CNN model can learn to extract more complicated characteristics unique to the dog breed classification problem with less training data by adjusting the network's upper levels.

Additionally, because the dataset utilized in this research is less than the original dataset used to train MobileNetV2, fine-tuning the parameters of the pre-trained network is essential to avoid overfitting. When a model performs well on the training set but badly on fresh, untainted data, overfitting has taken place. The network may learn the precise characteristics pertinent to the dog breed classification job while avoiding overfitting by fine-tuning the parameters of the pre-trained model.

In conclusion, by using the pre-trained MobileNetV2 architecture as a feature extractor and fine-tuning its higher layers to learn the specific features relevant to the task with less training data, the transfer learning approach can enhance the performance of the CNN model on the dog breed classification task.

3. SOFTMAX REGRESSION

The output layer of the CNN and transfer learning model for classifying dog breeds is built to contain as many neurons as there are dog breeds in the dataset. For instance, the output layer will have 120 neurons if the dataset contains 120 different dog breeds.

The output of the last fully connected layer is normalized during training by the Softmax activation function, producing a probability distribution over the dog breeds. The class in the output vector with the highest probability is the anticipated dog breed. The difference between the actual labels of the pictures in the training set and the expected probability distribution is assessed using the cross-entropy loss function.

EXPERIMENT

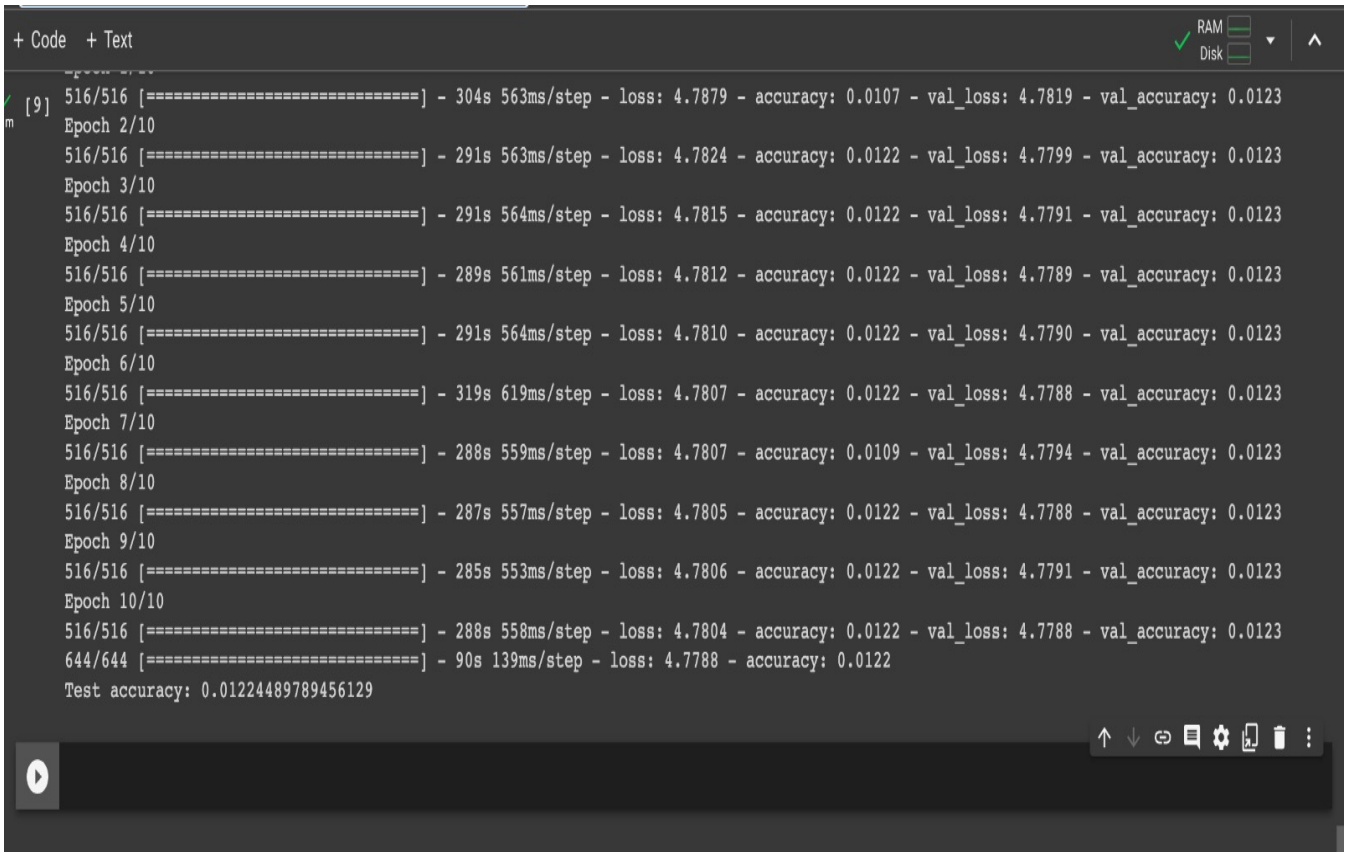
```
    class_names = os.listdir(data_dir)
    n_classes = len(class_names)

    test_ratio = test_ratio
    validation_ratio = validation_ratio

    train_filepaths = []
    valid_filepaths = []
    test_filepaths = []

    for i in range(n_classes):
        img_per_dog = os.listdir(os.path.join(data_dir, class_names[i]))
        img_per_dog = np.array([os.path.join(class_names[i], s) for s in img_per_dog])

        total_size = len(img_per_dog)
        test_size = int(total_size * test_ratio)
        validation_size = int(total_size * validation_ratio)
        train_size = total_size - test_size - validation_size

        rnd_indices = np.random.permutation(total_size)

        train_filepaths.append(img_per_dog[rnd_indices[:train_size]])
        valid_filepaths.append(img_per_dog[rnd_indices[train_size:-test_size]])
        test_filepaths.append(img_per_dog[rnd_indices[-test_size:]])

    train_filepaths = np.array([os.path.join(data_dir, s) for s in np.hstack(train_filepaths)])
    valid_filepaths = np.array([os.path.join(data_dir, s) for s in np.hstack(valid_filepaths)])
    test_filepaths = np.array([os.path.join(data_dir, s) for s in np.hstack(test_filepaths)])

    return (train_filepaths, valid_filepaths, test_filepaths)

(train_filepaths, valid_filepaths, test_filepaths) = filepaths(data_dir)
train_filepaths.shape, valid_filepaths.shape, test_filepaths.shape
```

[8]: ((12436,), (4072,), (4072,))

(Train&Test)size and path

```
    class_mode='categorical',
    shuffle=False
)

# Define the model architecture
input_shape = (224, 224, 3)
model = keras.Sequential(
    [
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(128, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(256, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),

        layers.Flatten(),
        layers.Dense(512, activation='relu'),
        layers.Dense(n_classes, activation='softmax')
    ]
)
```

```
 hNormalization)

 dropout (Dropout)           (None, 512)               0

 dense_3 (Dense)             (None, 120)               61560

=================================================================
Total params: 14,347,416
Trainable params: 14,344,472
Non-trainable params: 2,944

_________________________________________________________________
Epoch 1/50
516/516 [==============================] - 341s 640ms/step - loss: 5.3931 - accuracy: 0.0173 - val_loss: 4.9009 - val_accuracy: 0.0182
Epoch 2/50
516/516 [==============================] - 320s 620ms/step - loss: 4.9496 - accuracy: 0.0281 - val_loss: 4.6038 - val_accuracy: 0.0314
Epoch 3/50
516/516 [==============================] - 315s 611ms/step - loss: 4.6627 - accuracy: 0.0393 - val_loss: 4.5193 - val_accuracy: 0.0425
Epoch 4/50
516/516 [==============================] - 315s 611ms/step - loss: 4.4207 - accuracy: 0.0547 - val_loss: 4.1929 - val_accuracy: 0.0702
Epoch 5/50
516/516 [==============================] - 315s 611ms/step - loss: 4.2086 - accuracy: 0.0694 - val_loss: 4.3023 - val_accuracy: 0.0614
Epoch 6/50
```



In this design, convolutional layers are employed to extract significant information from the input picture. A number of filters are dragged over the input picture by each convolutional layer, producing feature maps. With each convolutional layer, there are more filters added. All convolutional layers employ ReLU as their activation function, which results in non-linearity in the output. The feature maps are down-sampled using max pooling layers, which decreases the output's spatial dimensions and aids in avoiding overfitting.

The above code defines a convolutional neural network (CNN) architecture that can be used for image classification tasks. Let me explain the different components of the code:

input_shape (224, 224, 3) specifies the size of the input images that the model expects. In this case, the input images are 224x224 pixels with 3 color channels (RGB).

model = keras.Sequential() initializes a sequential model, which allows you to stack layers on top of each other.

layers.Conv2D(32, (3, 3), activation-'relu', input_shape=input_shape) adds a 2D convolutional layer with 32 filters, a kernel size of (3, 3), and ReLU activation function. The input shape is specified as the input_shape argument.

layers.MaxPooling2D((2, 2)) adds a max pooling layer with a pool size of (2, 2) to reduce the spatial dimensions of the feature maps.

Steps 3 and 4 are repeated with increasing number of filters and decreasing feature map size in the subsequent convolutional layers: layers.Conv2D(64, (3, 3), activation='relu'), layers.MaxPooling2D((2, 2)), layers.Conv2D(128, (3, 3), activation='relu'), layers.MaxPooling2D((2, 2)), layers.Conv2D(256, (3, 3), activation='relu'), layers.MaxPooling2D((2, 2)).

layers.Flatten() flattens the output of the last convolutional layer into a 1-dimensional vector that can be fed into the fully connected layers.

layers.Dense(512, activation-'rel') adds a fully connected layer with 512 neurons and ReLU activation function.

layers.Dense(n_classes, activation='softmax') adds another fully connected layer with n_classes neurons (number of output categories) and softmax activation function, which produces a probability distribution over the output categories.

The speciality of this model architecture is that it has several convolutional layers with increasing number of filters and decreasing feature map size, followed by max pooling layers that reduce the spatial dimensions of the feature maps. This allows the model to learn hierarchical representations of the input images, capturing both low-level features (such as edges and corners) and high-level features (such as shapes and objects). The fully connected layers at the end of the model produce a probability distribution over the output categories, allowing the model to classify input images into one of the n_classes distinct categories.

With the layers mentioned ,it is not possible to achieve great results ,so we have included layers and changed the nodes ,filters,kernal size.(For better understanding refer the codes and their ouptuts in the results)

RESULTS

- We developed a classification model using the MobileNetV2 architecture after collecting the Stanford Dog Breeds Classification dataset and carrying out the appropriate preparation procedures described below.
- Regarding the input layers, the model architecture, and the number of parameters employed, the model summary offers insightful information.
- We used 40 epochs to train the model using the Transfer Learnng and 50 using the CNN , checking the accuracy, loss, and validation accuracy after each iteration.With the given patience the models stopped training where the improvement is negligable.
- We ran the model on the test data and computed the accuracy to assess its performance.
- These measures allowed us to assess the model's performance in precisely categorizing the dog breeds.

```
]: keras.backend.clear_session()
tf.random.set_seed(42)
np.random.seed(42)

base_model_mobilenet = tf.keras.applications.MobileNetV2(
                                    weights="imagenet", include_top=False)

# Create a new model on top
inputs = keras.Input(shape=(224, 224, 3))
x = data_augmentation(inputs)
x = base_model_mobilenet(x, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(512, activation='relu')(x)
x = layers.Dropout(0.3)(x)
outputs = layers.Dense(n_classes, activation="softmax")(x)
mobilenet_model = Model(inputs=inputs, outputs=outputs)

# Freeze the weights of the base model
for layer in base_model_mobilenet.layers:
    layer.trainable = False

# Compile the model and start training
mobilenet_model.compile(optimizer=keras.optimizers.Adam(lr=1e-4),
              loss=keras.losses.SparseCategoricalCrossentropy(),
              metrics=["accuracy"])
mobilenet_model.summary()

WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input sh
ape (224, 224) will be loaded as the default.

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optim
izers.legacy.Adam.

Model: "model"
```

The above image explains the Transfer Learning model design for Dog Breed Classification using the MobileNetV2.And the below image explains the summary of the model.

```
WARNING:tensorflow: Input_shape  is undefined or non-square, or rows  is not in [96, 128, 160, 192, 224]. Weights for input sh
ape (224, 224) will be loaded as the default.

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optim
izers.legacy.Adam.

Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 sequential (Sequential)     (None, 224, 224, 3)       0

 mobilenetv2_1.00_224 (Funct  (None, None, None, 1280)  2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 512)               655872

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 120)               61560

=================================================================
Total params: 2,975,416
Trainable params: 717,432
Non-trainable params: 2,257,984
_________________________________________________________________
```

```
In [ ]: epochs=40
        history_mobilenet = mobilenet_model.fit(train_set_mobilenet, epochs=epochs,
                                                validation_data=valid_set_mobilenet,
                                                callbacks=[early_stopping_cb, tensorboard_cb])
        # Let's save the model before we do fine-tuning
        mobilenet_model.save("mobilenet_without_fine_tuning.h5")
```

```
Epoch 1/40
389/389 [==============================] - 508s 1s/step - loss: 2.2753 - accuracy: 0.4285 - val_loss: 0.9764 - val_accuracy: 0.
7085
Epoch 2/40
389/389 [==============================] - 502s 1s/step - loss: 1.4447 - accuracy: 0.5891 - val_loss: 0.8435 - val_accuracy: 0.
7429
Epoch 3/40
389/389 [==============================] - 459s 1s/step - loss: 1.2713 - accuracy: 0.6265 - val_loss: 0.8508 - val_accuracy: 0.
7360
Epoch 4/40
389/389 [==============================] - 446s 1s/step - loss: 1.2072 - accuracy: 0.6469 - val_loss: 0.8450 - val_accuracy: 0.
7468
Epoch 5/40
389/389 [==============================] - 466s 1s/step - loss: 1.1392 - accuracy: 0.6632 - val_loss: 0.8138 - val_accuracy: 0.
7446
Epoch 6/40
389/389 [==============================] - 467s 1s/step - loss: 1.0889 - accuracy: 0.6781 - val_loss: 0.8248 - val_accuracy: 0.
7502
Epoch 7/40
389/389 [==============================] - 461s 1s/step - loss: 1.0301 - accuracy: 0.6954 - val_loss: 0.8295 - val_accuracy: 0.
7448
Epoch 8/40
389/389 [==============================] - 467s 1s/step - loss: 0.9958 - accuracy: 0.6993 - val_loss: 0.7972 - val_accuracy: 0.
7662
Epoch 9/40
389/389 [==============================] - 467s 1s/step - loss: 0.9684 - accuracy: 0.7075 - val_loss: 0.8200 - val_accuracy: 0.
7534
Epoch 10/40
389/389 [==============================] - 469s 1s/step - loss: 0.9337 - accuracy: 0.7164 - val_loss: 0.8366 - val_accuracy: 0.
```

TL Epochs

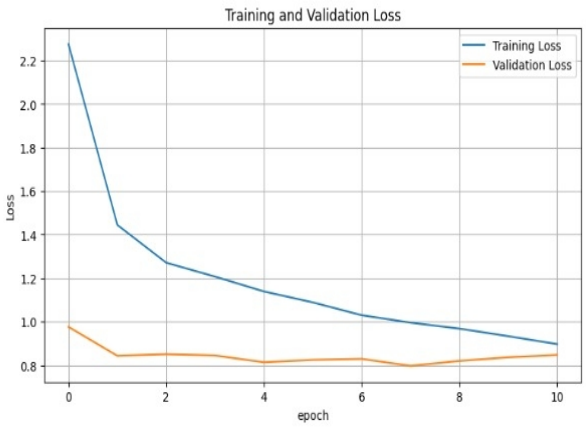
```
    [
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        layers.BatchNormalization(),
        layers.Conv2D(32, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(128, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.Conv2D(128, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(256, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.Conv2D(256, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),

        layers.Flatten(),
        layers.Dense(512, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.5),
        layers.Dense(n_classes, activation='softmax')
    ]
```

The above image explains the CNN model which is built from the beginning i.e adjusting the weights and following back propogation,where 8 convolution layers have been used unlike in the experiments section with the different nodes and filter,kernal sizes

```
Epoch 33/50
516/516 [==============================] - 322s 624ms/step - loss: 1.6957 - accuracy: 0.5165 - val_loss: 2.1325 - val_accurac
y: 0.4315
Epoch 34/50
516/516 [==============================] - 316s 612ms/step - loss: 1.6605 - accuracy: 0.5265 - val_loss: 2.0641 - val_accurac
y: 0.4371
Epoch 35/50
516/516 [==============================] - 315s 610ms/step - loss: 1.6032 - accuracy: 0.5436 - val_loss: 2.2915 - val_accurac
y: 0.4128
Epoch 36/50
516/516 [==============================] - 317s 615ms/step - loss: 1.5860 - accuracy: 0.5486 - val_loss: 2.0865 - val_accurac
y: 0.4487
Epoch 37/50
516/516 [==============================] - 315s 611ms/step - loss: 1.5681 - accuracy: 0.5489 - val_loss: 2.2260 - val_accurac
y: 0.4334
644/644 [==============================] - 96s 148ms/step - loss: 1.5065 - accuracy: 0.5760
Test accuracy: 0.5759961009025574
```

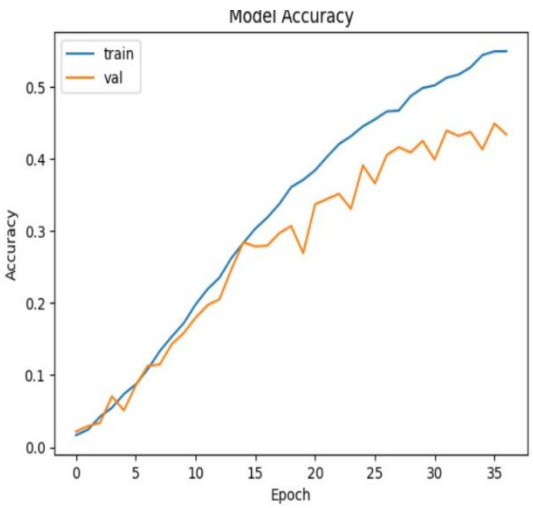
CNN Epochs

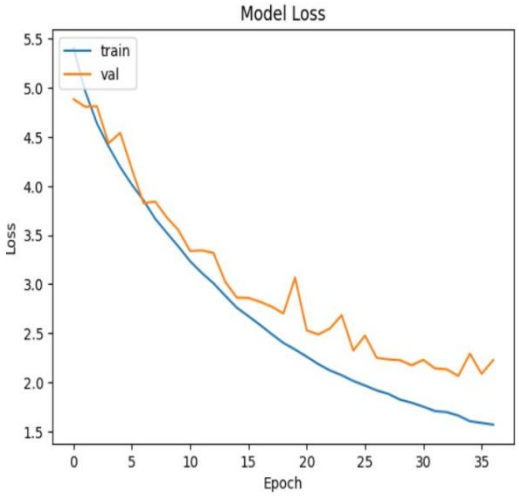
```
plot_learning_curves(history_mobilenet)
```




```
In [12]: # Evaluate the final model on the test set
```

TL Training Validation Accuracy and Loss





CNN Training Validation Accuracy and Loss

epoch epoch

```
[12]: # Evaluate the final model on the test set
      final_model = keras.models.load_model("mobilenet_without_fine_tuning.h5")
      loss, accuracy = final_model.evaluate(test_set_mobilenet)
      print('Test accuracy :', np.round((accuracy * 100), 2), '%')

      128/128 [==============================] - 147s 1s/step - loss: 0.5952 - accuracy: 0.8158
      Test accuracy : 81.58 %
```

TL Test_accuracy

```
y: 0.4487
Epoch 37/50
516/516 [==============================] - 315s 611ms/step - loss: 1.5681 - accuracy: 0.5489 - val_loss: 2.2260 - val_accurac
y: 0.4334
644/644 [==============================] - 96s 148ms/step - loss: 1.5065 - accuracy: 0.5760
Test accuracy: 0.5759961009025574
```

Image 7 : CNN Test_accuracy

The results obtained by both transfer learning and the CNN model created from scratch for the specified epochs and the executed pre processing accuracies are explained in the above images.We can classify dog breeds using transfer learning based on accuracy.Additionally, it was discovered that transfer learning had substantially higher accuracy levels than the CNN model created for classification purposes.Despite being designed for different picture categorization, transfer learning worked well and produced predictions faster than the CNN model.By preprocessing data and adding more data, CNN's accuracy can be increased.Finally, the findings help us to comprehend the significance and value of transfer learning.

CONCLUSION

- According to the project's findings, transfer learning can be a very efficient method for enhancing the precision and efficiency of deep learning models for image categorization tasks. With a lower size and shorter training period, the MobileNetV2 model outperformed the CNN model created from scratch in terms of accuracy.

- As transfer learning can save time and money while obtaining cutting-edge performance, this result can be helpful for academics and practitioners in the field of computer vision who are aiming to construct high-performance models for image classification tasks.

- The study also emphasizes the significance of cleaning and preparing data to enhance model performance. The quantity and quality of data are important variables that can have a considerable impact on the precision and generalization of the models, therefore this result can be valuable for anybody working with machine learning and deep learning models.

- Overall, the project has the potential to progress AI research and development by offering insights into efficient deep learning methodologies and techniques as well as by highlighting the potential of AI to address real-world issues and enhance societal quality of life.

Future Plans

A CNN model built from scratch and a transfer learning method leveraging MobileNetV2 were required for the assignment, which involved developing and comparing two deep learning models for supervised learning-based dog breed categorization. The goal was to evaluate the costs and benefits of building a model from start vs using one that has previously been trained, as well as the value of transfer learning in boosting model accuracy.

The project required data collection, preprocessing, model training, model assessment, and outcome analysis. Overall, the project provided an opportunity for participants to research, learn about, and apply a variety of deep learning approaches and techniques to a challenging and interesting computer vision problem.

Convolutional neural networks (CNNs) with attention mechanisms, generative adversarial networks (GANs), and reinforcement learning algorithms are some of the more sophisticated deep learning approaches and models that I would like to investigate and play with if I had more time.

In addition, we like to concentrate on creating useful deep learning applications, such as object identification and image recognition for autonomous vehicles, analysis of medical imaging, and natural language processing for chatbots and virtual assistants.

SUMMARY

The study comprises developing and contrasting two deep learning models for supervised learning dog breed classification. The first model is a CNN model that was created from the ground up with the goal of categorizing dog breeds in mind. The second model uses MobileNetV2, a pre-trained model that has previously been trained on a sizable dataset of photos, as part of a transfer learning strategy.

The goal of the study is to evaluate the accuracy of these two models as well as the quantity of data and computing power needed to train them. The study specifically aims to examine the trade-offs involved in creating a model from scratch as opposed to utilizing a pre-trained model and to ascertain the efficiency of transfer learning in enhancing the accuracy of the model. Overall, the project offers a fantastic chance to investigate and learn about various deep learning techniques and approaches, as well as to apply them to a tough and engaging computer vision issue.

The task entailed creating and contrasting two deep learning models for supervised learning-based dog breed classification: a CNN model created from scratch and a transfer learning strategy utilizing MobileNetV2. The objective was to assess the trade-offs between creating a model from scratch and adopting one that has already been trained, as well as the usefulness of transfer learning in increasing model accuracy. Data collection, preprocessing, model training, model evaluation, and outcome analysis were all necessary for the project. Overall, the project gave participants the chance to investigate, learn about, and apply several deep learning methods and techniques to a tough and engaging computer vision problem.

REFERENCES

1. https://www.image-net.org/
2. https://www.kaggle.com/code/yacinerouizi/stanford-dogs-dataset-part-2-transfer-learning

3. https://www.tensorflow.org/tutorials/images/transfer_learning

4. https://www.kaggle.com/code/friskycodeur/image-recognition-using-cnn-explained

4. Chauhan, Rahul; Ghanshala, Kamal Kumar; Joshi, R.C (2018). [IEEE 2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC) - Jalandhar, India (2018.12.15-2018.12.17)] 2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC) - Convolutional Neural Network (CNN) for Image Detection and Recognition. , (), 278–282.

5. http://vision.stanford.edu/aditya86/ImageNetDogs/

6. https://www.tensorflow.org/tutorials/images/transfer_learning

7. https://www.tensorflow.org/hub