

Deep learning based human action recognition

Sanketh Bennur

Submitted for the Degree of Master of Science in
Data Science and Analytics



Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

September 5, 2022

Declaration

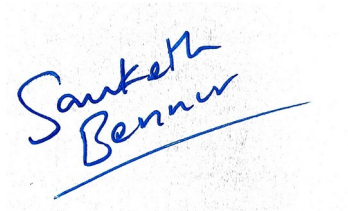
This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 12,550

Student Name: Sanketh Bennur

Date of Submission: 5 September 2022

Signature:



Acknowledgements

I would like to thank my supervisor Dr Li Zhang for her support and advice throughout this work. She is very knowledgeable and helpful. She acquainted and enlightened me with knowledge on the many concepts of this domain. Her guidance led me to put in lot of effort and passion into this work.

Abstract

Computer Vision and Machine Learning is being used widely in several domains and especially for Human Action Recognition. With the help of Image Processing and Artificial Neural Networks, Human Action Recognition is more easily achievable. Transfer Learning involves using stored knowledge and applying the same to a different problem in the same domain.

In the field of Machine Learning, Transfer Learning of Neural Networks involves using a trained Neural Network Machine Learning model to be trained again on a new dataset, so that the model can be used to infer on newer related dataset. The proposed work compares various pre-trained Neural Network models such as VGG16, ResNet50, Inception-V3 and MobileNet for transfer learning in Human Action Recognition. Transfer Learning have been performed on the above models and the models have been compared on the factors – accuracy, loss, storage size, number of parameters, architecture etc.

Accuracy of a machine learning model is the percentage of predictions that were correct for any given test data. The Loss of a machine learning model is the percentage of error in model's prediction for a particular test data. The output obtained during the training of the models is used and graphs of Accuracy and Loss of the models are plotted. They accuracy, loss and training times for each of the models for different number of classifications are observed – 5, 10, 15 and 20 action classes from the UCF-101 dataset.

Index Terms - Transfer Learning, Deep Learning Architectures, Inception-v3, Resnet50, MobileNet, VGG-16, Neurons, Layers, Tensorflow, Keras

Contents

1	Introduction	6
2	Related Work	9
2.1	Inception-V3	11
2.2	ResNet-50	12
2.3	MobileNet	13
2.4	VGG-16	14
3	Proposed Work	15
3.1	Implementation	15
3.1.1	Gathering the data	16
3.1.2	Training the models	19
3.2	Inception-V3 Transfer Learning	26
3.3	MobileNet Transfer Learning	27
3.4	ResNet-50 Transfer Learning	28
3.5	VGG-16 Transfer Learning	30
4	Evaluation	33
4.1	Inception-V3 Results	33
4.1.1	5 Action Classes	33
4.1.2	10 Action Classes.....	36
4.1.3	15 Action Classes	37
4.1.4	20 Action Classes	39
4.2	MobileNet Results	42
4.2.1	5 Action Classes	42
4.2.2	10 Action Classes.....	44
4.2.3	15 Action Classes	46
4.2.4	20 Action Classes	48
4.3	ResNet-50 Results	50
4.3.1	5 Action Classes	51

4.3.2	10 Action Classes.....	54
4.3.3	15 Action Classes	56
4.3.4	20 Action Classes	58
4.4	VGG-16 Results	60
4.4.1	5 Action Classes	61
4.4.2	10 Action Classes.....	63
4.4.3	15 Action Classes	65
4.4.4	20 Action Classes	67
5	Conclusion and Future Work	69
6	Self-Examination	72
7	Professional Issues	74
8	References	76
9	Appendix	80
9.1	Using the Code	80
9.2	Code	84
9.1.1	Extract Frames from Videos	84
9.1.2	Train and Evaluate Machine Learning	85
9.1.3	Models stored in Hard-Drive	89

1 Introduction

Human Action Recognition involves having to recognize the physical actions performed by humans in various fields such as sports, athletics etc. Humans can recognise actions in real time with the help of sophisticated sensory inputs and access to a wealth of contextual information. Machines are different in that regard. Which data points in the image correspond to the object of interest must be considered for the machine to accurately identify the activity [5]. Tasks involving video action recognition necessitate the addition of a dimension that represents temporal interdependence. For videos, frames can be extracted from videos for Image Classification.

The use of computer information technology in numerous industries has grown increasingly widespread as a result of the ongoing growth of society and the economy as well as the ongoing progress of science and technology. Technology of this nature can help businesses produce goods of a high standard and to a certain extent, increase their production efficiency. Consequently, in order to provide a specific reference, this study focuses on computer vision technology in Deep Learning based Human Action Recognition.

Artificial intelligence is a field of study that enables machines to solve intuitive issues by mimicking how the human brain functions. The recent development of artificial intelligence-based components has strengthened computer vision research in fields like digital image processing, object recognition, and object tracking [1]. Consequently, findings can be acquired more quickly and accurately.

Computer vision is a branch of computer science that focuses on simulating some of the complexity of the human visual system and enabling computers to recognise and analyze items in pictures and videos in a similar manner to humans [3]. Computer vision has only recently begun to function more fully. In several tasks linked to object detection and categorization, the field has made significant strides in recent years and has been able to outperform humans.

Deep Learning involves having to use Artificial Neural Network. Deep Learning models like Inception-v3, Resnet50, MobileNet and VGG-16 have several convolutions and maxpool layers. These layers involve fully connected neurons in each layer that receive inputs and send a value as an output [4][13]. In this manner, the neurons are 'activated' similar to the functioning of the human brain and can predict among different classification categories as output.

Transfer learning is a machine learning technique where a machine learning model, usually a Deep Learning Artificial Neural Network model, created for one job is utilized as the foundation for a model for another task [8]. Pre-trained models are frequently used as the foundation for deep learning tasks in computer vision and natural language processing due to the enormous time and computing resources needed to develop neural network models for these problems as well as the enormous gains in performance they offer on related problems [14].

Inception-V3 is a convolutional neural network developed using training data from the ImageNet collection, which contains more than one million images [18]. For use in mobile and embedded vision applications, MobileNet is a particular kind of convolutional neural network, which is based on a simplified design that employs depth-wise separable convolutions to construct compact deep neural networks that can have minimal latency for embedded and mobile devices [22].

ResNet, which stands for Residual Networks, is a well-known neural network that serves as the foundation for many computer vision applications. ResNet's core innovation was that it enabled us to successfully train incredibly deep neural networks with more than 150 layers. Prior to ResNet, the issue of vanishing gradients made it challenging to train very deep neural networks [15].

TensorFlow's first mobile computer vision model, MobileNet is primarily used in mobile apps. Convolutions that can be separated based on depth are used by MobileNet. When compared to a network with conventional convolutions of the same depth in the nets, it dramatically lowers the number of parameters. Deep neural networks are hence lightweight. [26].

VGG16 is another famous Deep Learning Artificial Neural Network model. The 16 in VGG16 denotes that it contains 16 layers with weights. This network has 138 million parameters, making it a fairly sizeable network [30]. Convolution and max pool layers are arranged throughout the entire architecture. In the end, it has two fully connected layers (FC), which are followed by a softmax for output.

In this work, a comprehensive study has been made on many pre-trained Deep Learning Artificial Neural Network models and Transfer Learning. A comparison is made on the models Inception-v3, Resnet50, MobileNet and VGG-16 on the factors - Accuracy, Loss, Training Time, Storage Size, number of parameters etc.

The difference between the outcome produced by the machine learning model and the intended outcome is measured by the loss function in a neural network. The parameters are the values that has to be tuned to get good model results by optimising a loss function. In the case of Neural Networks, the number of parameters is the measurement of the neural networks, comprising of layers in the network and the number of units in each layer. In this study, the number of parameters of each model is also compared to study the density and accuracies of the models.

Transfer Learning has been used to train the pre-trained models with another dataset by adding newer Fully Connected Layers. These models have been evaluated on a test dataset and the performance of these models have been recorded. The models have also been saved in storage and their size, accuracy, loss and training time have been compared.

2 Related Work

A subset of machine learning techniques, called deep learning, seeks to identify several levels of distributed representations. Numerous deep learning techniques have recently been presented to address common artificial intelligence issues. By highlighting the contributions and difficulties from recent research papers, this study seeks to examine the state-of-the-art deep learning methods for computer vision. It provides an overview of different deep learning techniques and recent breakthroughs before briefly describing how they might be used for a variety of visual tasks [5]. It primarily focuses on Image Recognition using Deep Learning.

Applications for deep learning frequently employ transfer learning. Transfer learning usually makes it far quicker and simpler to fine-tune a network than to train one from scratch using randomly initialized weights. A lower number of training photos can be used to swiftly apply learnt features to a new assignment. In this study, multiclass picture categorization is implemented using ultra-deep neural networks trained on the ResNet50 architectural network. The anti-aliasing approach is used by the normalizing algorithm to create the resulting greyscale images, which have a 99% recognition rate compared to the slower machine learning approaches such as KNN or DNN [13].

The performance of transfer learning in convolutional neural networks depends on the selection of which layers are to be learned again and which are not [8]. Transfer learning gives CNNs the opportunity to learn with small data samples by transferring knowledge from models that have already been trained on extensive amounts of data. The following work suggested attentive feature distillation and selection (AFDS), which not only modifies the strength of regularization of transfer learning but also dynamically selects the critical features to transfer. The research claim that by using AFDS on ResNet-101, a cutting-edge computation reduction has been made possible at the same accuracy budget, exceeding all currently used transfer learning techniques [8].

The research demonstrated in this study suggests that an attacker can launch an effective and efficient brute force attack that can create instances of input to trigger each target class with high confidence without any further knowledge outside of the pre-trained model. The researchers ran a series of trials on tasks including speech and facial recognition to gauge the effectiveness of the suggested approach [9]. For the task of classifying maritime vessels using the MARVEL dataset, transfer learning parameters are explored on the AlexNet, VGGNet, and

ResNet architectures. The outcomes shown that fine-tuning the other layers, whose weights are initialised from pre-trained weights, outperforms training the network from scratch [9].

The robustness of transfer learning models to adversarial attacks is investigated. A resilient ImageNet source model is transferred onto the CIFAR domain to show the effectiveness of robust transfer learning. A robust CIFAR-100 model is built and the generalization of it is increased by about 2% while maintaining its robustness. [35] Since there is no fixed way for choosing layers that will perform as well as possible, we created the Differential Evolution based Fine-Tuning (DEFT) method to choose fine-tunable layers for a target dataset while taking into account the restrictions. The technique was tested against the challenge of locating the osteosarcoma in the medical imaging dataset.

The following work suggests a novel mathematical approach called Deep Transfer Learning By Exploring Where To Transfer (DT-LET) to address this heterogeneous transfer learning issue. Researchers established a special loss function to measure the correspondence between high-level data features in the source domain and the target domain in order to choose the best matching of layers to transfer knowledge with the test data. The research described deep transfer learning, classified it, and reviewed recent research studies that used deep transfer learning approaches. [12]

The Ladder network provides the foundation for this research's work, which is expanded by fusing the model with supervision. The study demonstrated that the resulting model achieves state-of-the-art performance in a variety of tasks, including MNIST and CIFAR-10 classification in a semi-supervised scenario as well as permutation invariant MNIST in both settings. The findings of this study give a general overview of the application of transfer learning methods to masked face recognition and comparison between the MobileNetV2 and VGG16 models. With an accuracy rating of 95.42 percent, the results demonstrated that the MobileNetV2 model was superior to VGG16. [4]

2.1 InceptionV3

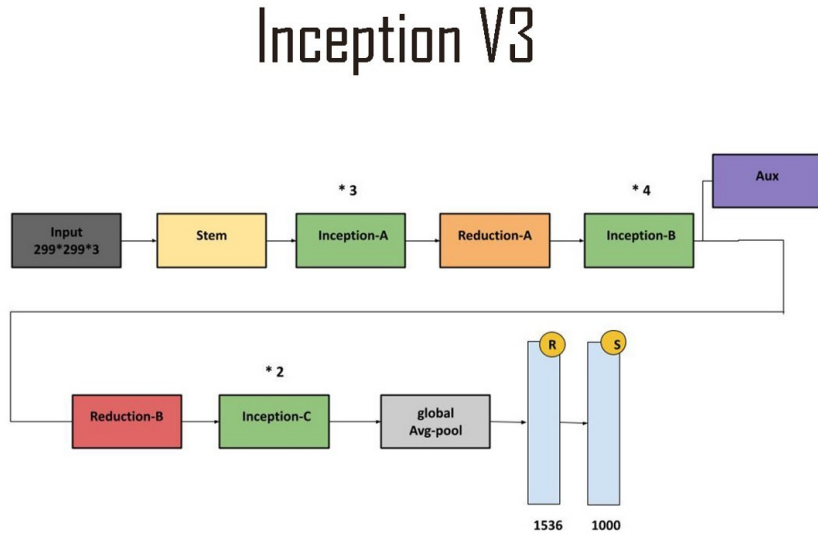


Figure 1: Inception-V3 architecture

(source: https://production-media.paperswithcode.com/methods/inceptionv3onc--oview_vjAbOfw.png)

An image recognition model called Inception-V3 has demonstrated higher than 78.1% accuracy on the ImageNet dataset. Convolutions, average pooling, max pooling, concatenations, dropouts, and fully connected layers are some of the symmetric and asymmetric building elements used in InceptionV3. Inputs for activation are subjected to batch normalisation, which is utilised widely across the model. Softmax is used to calculate loss. [20]

InceptionV3, which has already been trained, is tweaked to create the proposed model in order to detect face masks for monitoring during COVID – 19. By obtaining an accuracy of 99.9% during training and 100% during testing, the model surpassed the other recently proposed methods [19]. In [21] pulmonary image data is enhanced, the features are extracted automatically using the fine-tuned Inception-v3 model based on transfer learning, and classified the pulmonary pictures using several classifiers, showing maximum sensitivity and specificity are 95.41% and 80.09%, respectively, and it performed better than other methods for classifying pulmonary images.

2.2 Resnet-50

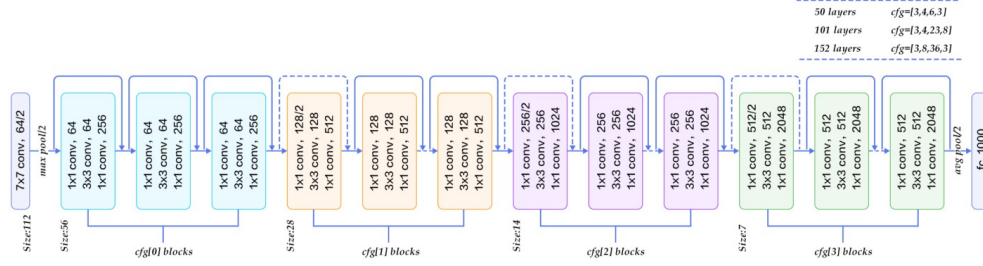


Figure 2: ResNet-50 architecture

(source: https://miro.medium.com/max/1400/0*9LqUp7XyEx1QNc6A.png)

ResNet50 is a ResNet model variant having 48 Convolution layers, 1 MaxPool layer, and 1 Average Pool layer. It can operate with floating point numbers (3.8×10^9). The network has picked up detailed feature representations for many different types of images. The network can accept images up to 224 by 224. [14]

In [13] a DCNN based on Resnet-50 is developed, which can distinguish COVID-19 from two other classes (pneumonia and normal) in chest X-ray pictures. Two classification techniques, multi-class (BC-2: COVID-19 vs. pneumonia) and binary (BC-1: COVID-19 vs. normal), were used to assess DCNN (pneumonia vs. normal vs. COVID-19). The average accuracy of this design is 99.9% for BC-1 instances, 99.8% for BC-2 cases, and 97.3% for multiclass cases. In order to increase the diagnostic precision, the research in [15] focuses on the application of transfer learning-based classification of cells infected with malaria. The outcomes of the experiment demonstrate that transfer learning works well with tiny cell images.

2.3 MobileNet

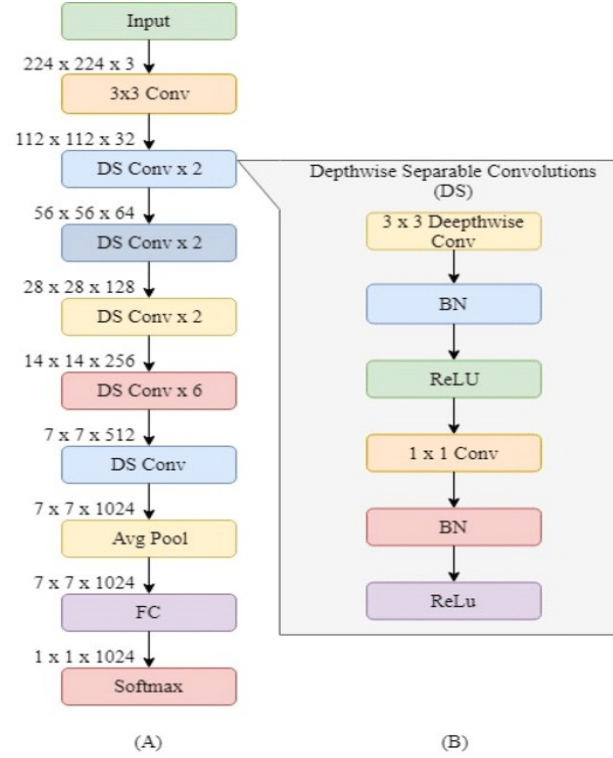


Figure 3: MobileNet Architecture

(Source: [https://www.researchgate.net/profile/Olarik-Surinta/](https://www.researchgate.net/profile/Olarik-Surinta/publication/340470168/figure/fig3/AS:877502188769280@1586224233016/)

publication/340470168/figure/fig3/AS:877502188769280@1586224233016/

Illustration-of-the-MobileNet-architecture-A-The-overall-MobileNet-architecture-and.png)

The first mobile computer vision model made by TensorFlow is the MobileNet. It finds its use in mobile apps. Depth-wise separable convolutions are used by MobileNet. When compared to a network with conventional convolutions of the same depth in the nets, it dramatically reduces the number of parameters. Lightweight deep neural networks are the outcome of this. [25] This offers us a great place to start when training our classifiers, which are ridiculously small and unbelievably quick. MobileNet is a class of CNN that was open-sourced by Google.

Deep learning models have attracted attention for their potential to improve the efficiency of computer-aided diagnosis systems in mammography (CADx). Resnet50 and MobileNet produced the best outcomes, scoring 78.4% and

74.3%, respectively [23]. The various real-time applications of MobileNet is discussed in [24]-[27]. Only 40M in size, the MobilenetV2 model is ideal for embedding in marine aquaculture systems and classifying photos of marine animals. When compared to MobileNet without transfer learning, MobileNet with transfer learning achieves 95.63% accuracy in [26].

2.4 VGG-16

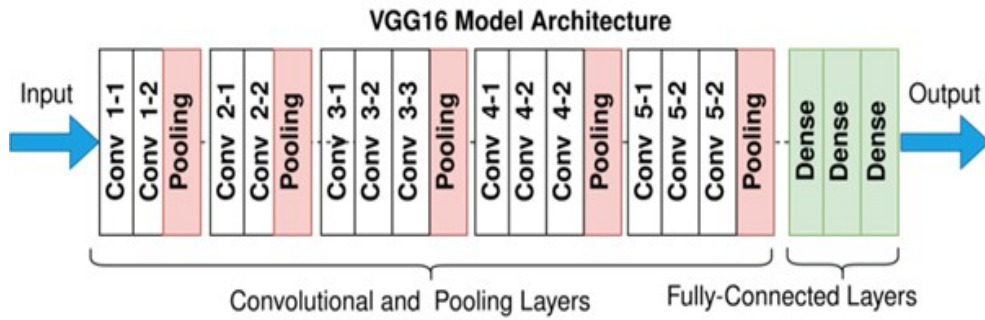


Figure 4: VGG-16 architecture

(source: <https://upload.wikimedia.org/wikipedia/commons/thumb/2/24/VGG16.png/1280px-VGG16.png>)

The University of Oxford professors K. Simonyan and A. Zisserman devised the convolutional neural network model known as VGG16. In the top five tests, the model performs 92.7% accurately in ImageNet, a dataset of over 14 million images divided into 1000 classes. It was a well-known model that was submitted to ILSVRC-2014. By sequentially substituting several 3x3 kernel-sized filters for AlexNet's big kernel-sized filters (11 and 5, respectively, in the first and second convolutional layers), it improves upon AlexNet. Using NVIDIA Titan Black GPUs, VGG16 underwent weeks of training. [30]

Transfer learning is utilised in this study to optimise the parameters of the pre-trained network (VGG19) for the image classification task and is compared with AlexNet and VGG16. Along with the CNN architectures, a hybrid learning strategy that compares support vector machine (SVM) classifier after robust feature extraction from CNN architecture is used. Performance research demonstrates that for the purpose of image classification, the tuned VGG19 architecture outperforms the other CNN and hybrid learning approaches [29]. The research in [28] and [30] discuss the Alzheimer's and Brain Image classification with respect to VGG16, which outperforms other deep-learning based methods.

3 Proposed Work

Transfer Learning has been used to train the pre-trained models with another dataset by adding newer Fully Connected Layers. In the models to be used for Transfer Learning, the weights that were pre-trained and set in each unit in each layer must be preserved. All layers of the model need not be used for training again during Transfer Learning on a new dataset.

3.1 Implementation

In the proposed work, Python language is used for scripting and Tensorflow library is used to build the models required for Transfer Learning in Jupyter Notebook environment. Tensorflow is an open-source library used for Machine Learning, with which several Machine Learning models can be built, trained and deployed for real-world applications. 'matplotlib' library is used for creating the plot to visualize Accuracy values and Loss values during training of the models.

Tensorflow Library already has the pre-trained models Inception-v3, Resnet50, MobileNet and VGG-16. We can call and initialize these models with the Tensorflow library. The wrapper classes of these models can be imported. In order to speeden up the training, train for several iterations, reduce the load on the processor, developers can choose to use Graphical Processing Unit in their machines for this purpose. Certain Nvidia GPU models come with CUDA cores that are cores in the processor that are dedicated to Deep Learning – training, evaluation, prediction, usage etc.

```
In [1]: from keras import models
        from keras.layers import Dense, Flatten
        from keras import backend as K
        import numpy as np
        import matplotlib.pyplot as plt

        from keras.applications import inception_v3

In [2]: import tensorflow as tf
        print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU'))
Num GPUs Available: 1
```

3.1.1 Gathering the Data

UCF101 dataset is used for the Transfer Learning of these models. UCF101 is a collection of real-world action videos from YouTube that have been categorized into 101 different action types. The most difficult data set to date, UCF101 has the greatest variety of actions and has the widest range of camera motion, object size, object scale, object perspective, backdrop clutter, lighting conditions, and more. The dataset can be found here:

<https://www.crcv.ucf.edu/research/data-sets/ucf101/>



Figure 5: UCF-101 Dataset action classes

(source: catelyzex.com, <https://ai2-s2-public.s3.amazonaws.com/figures/2017-08-08/b5f2846a506fc417e7da43f6a7679146d99c5e96/2-Figure1-1.png>)

The models are tested for different number of action classes for classification - 5, 10, 15, 20 classes. The Machine Learning models require images for training, evaluation and prediction. Hence, frames are extracted from the videos of their respective action classes. For the proposed work, an optimum number of frames of 60 of each video is chosen, since more frames can lead to underfitting and less frames can lead to overfitting. [41]


```
In [1]: # Importing all necessary libraries
import cv2
import os
```

```
In [2]: def get_frames(sourcedir, targetdir):
    if not os.path.exists(targetdir): os.mkdir(targetdir)

    for file in os.listdir(sourcedir):
        cap = cv2.VideoCapture(sourcedir+file)
        i = 0
        # a variable to set how many frames you want to skip
        frame_skip = 60
        # a variable to keep track of the frame to be saved
        frame_count = 0
        while cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                break
            if i > frame_skip - 1:
                frame_count += 1
                # cv2.imwrite('./frames/'+filename+str(frame_count*frame_skip)+'.jpg', frame)
                cv2.imwrite(targetdir+file+str(frame_count*frame_skip)+'.jpg', frame)
                i = 0
                continue
            i += 1

    cap.release()
    cv2.destroyAllWindows()
```

```
In [3]: get_frames("./UCF101-videos/ApplyLipstick/", "./frames/ApplyLipstick/")
get_frames("./UCF101-videos/Archery/", "./frames/Archery/")
get_frames("./UCF101-videos/BabyCrawling/", "./frames/BabyCrawling/")
get_frames("./UCF101-videos/Basketball/", "./frames/Basketball/")

get_frames("./UCF101-videos/Biking/", "./frames/Biking/")
get_frames("./UCF101-videos/Diving/", "./frames/Diving/")
get_frames("./UCF101-videos/Fencing/", "./frames/Fencing/")
get_frames("./UCF101-videos/IceDancing/", "./frames/IceDancing/")

get_frames("./UCF101-videos/Kayaking/", "./frames/Kayaking/")
get_frames("./UCF101-videos/MilitaryParade/", "./frames/MilitaryParade/")
get_frames("./UCF101-videos/PizzaTossing/", "./frames/PizzaTossing/")
get_frames("./UCF101-videos/PullUps/", "./frames/PullUps/")

get_frames("./UCF101-videos/ShavingBeard/", "./frames/ShavingBeard/")
get_frames("./UCF101-videos/SkateBoarding/", "./frames/SkateBoarding/")
get_frames("./UCF101-videos/SumoWrestling/", "./frames/SumoWrestling/")
get_frames("./UCF101-videos/Surfing/", "./frames/Surfing/")

get_frames("./UCF101-videos/TennisSwing/", "./frames/TennisSwing/")
get_frames("./UCF101-videos/Typing/", "./frames/Typing/")
get_frames("./UCF101-videos/WritingOnBoard/", "./frames/WritingOnBoard/")
get_frames("./UCF101-videos/YoYo/", "./frames/YoYo/")
```

Using Tensorflow's ImageDataGenerator instance, a split can be created for training images, validation images and test images. Using the 'flow_from_directory' function, a validation split of 20% is made. So 80% of the images as frames of the videos are used as training images.

```
In [3]: from keras.preprocessing.image import ImageDataGenerator

dataset_path = "./frames/"
# will contain the categories in respective folders

# Data generators
train_datagen = ImageDataGenerator(rescale=1/255, validation_split=0.2)
```

```
In [4]: image_size = (299,299)
        batch_size = 10

        train_batches = train_datagen.flow_from_directory(
            dataset_path,
            target_size = image_size,
            batch_size = batch_size,
            class_mode = "categorical",
            subset = "training"
        )

        validation_batches = train_datagen.flow_from_directory(
            dataset_path,
            target_size = image_size,
            batch_size = batch_size,
            class_mode = "categorical",
            subset = "validation"
        )

        test_batches = train_datagen.flow_from_directory(
            dataset_path,
            target_size = image_size,
            batch_size = batch_size,
            class_mode = "categorical",
            subset = "validation"
        )

Found 1546 images belonging to 5 classes.
Found 384 images belonging to 5 classes.
Found 384 images belonging to 5 classes.
```

The following classes have been used for training:

```
In [5]: train_batches.class_indices

Out[5]: {'ApplyLipstick': 0,
        'Archery': 1,
        'BabyCrawling': 2,
        'Basketball': 3,
        'Biking': 4,
        'Diving': 5,
        'Fencing': 6,
        'IceDancing': 7,
        'Kayaking': 8,
        'MilitaryParade': 9,
        'PizzaTossing': 10,
        'PullUps': 11,
        'ShavingBeard': 12,
        'SkateBoarding': 13,
        'SumoWrestling': 14,
        'Surfing': 15,
        'TennisSwing': 16,
        'Typing': 17,
        'WritingOnBoard': 18,
        'YoYo': 19}
```

Random images can be displayed and their classes can be seen. In this case, the classes are stored as a List of size 20. Classes are 0 or 1 in the range of 0 to 19.

```
In [6]: from matplotlib import pyplot as plt

def plot_images(images_arr):
    fig, axes = plt.subplots(1,10)
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()
```

```
In [7]: imgs, labels = train_batches[0]
plot_images(imgs)
print(labels[:10])
```



```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

The 1s represent the image in the data folder of the indexed class (between 0 – 19). The output above shows 1 for the image procured from that directory of the class. This is called *One-Hot Encoding*.

After gathering the data and being able to call the models into the development environment, the next stage of the methodology is to make and compile the models for Transfer Learning and ready for training on a new dataset.

3.1.2 Training the models

In the models to be used for Transfer Learning, the weights that were pre-trained and set in each unit in each layer must be preserved. All layers of the model need not be used for training again during Transfer Learning on a new dataset.

The models have certain parameters. 'weights' to specify which dataset was used to set weights for the pre-trained models - in this case weights configured by training on 'imagenet' dataset it used, 'classes' to specify number of classes for the use-case, a flag variable 'include_top' to include or not the last few layers usually used for fine-tuning and feature extraction.

```
In [8]: vggmodeltop = vgg16.VGG16(include_top=True,
                                input_shape=(224,224,3),
                                pooling='avg',
                                weights='imagenet')

for (i,layer) in enumerate(vggmodeltop.layers):
    layer.trainable = False
    print((i, layer.name, layer.output_shape))
```

```
(0, 'input_1', [(None, 224, 224, 3)])
(1, 'block1_conv1', (None, 224, 224, 64))
(2, 'block1_conv2', (None, 224, 224, 64))
(3, 'block1_pool', (None, 112, 112, 64))
(4, 'block2_conv1', (None, 112, 112, 128))
(5, 'block2_conv2', (None, 112, 112, 128))
(6, 'block2_pool', (None, 56, 56, 128))
(7, 'block3_conv1', (None, 56, 56, 256))
(8, 'block3_conv2', (None, 56, 56, 256))
(9, 'block3_conv3', (None, 56, 56, 256))
(10, 'block3_pool', (None, 28, 28, 256))
(11, 'block4_conv1', (None, 28, 28, 512))
(12, 'block4_conv2', (None, 28, 28, 512))
(13, 'block4_conv3', (None, 28, 28, 512))
(14, 'block4_pool', (None, 14, 14, 512))
(15, 'block5_conv1', (None, 14, 14, 512))
(16, 'block5_conv2', (None, 14, 14, 512))
(17, 'block5_conv3', (None, 14, 14, 512))
(18, 'block5_pool', (None, 7, 7, 512))
(19, 'flatten', (None, 25088))
(20, 'fc1', (None, 4096))
(21, 'fc2', (None, 4096))
(22, 'predictions', (None, 1000))
```

The models are called with the parameter 'include_top' set to False. The model will be returned without the last few layers, which are ideally used for fine-tuning and feature extraction. In the proposed work, the model containing last few hidden layers are also called to make a comparison with the model without last few hidden layers. The difference in number of last layers are observed.

```

In [9]: vggmodel = vgg16.VGG16(include_top=False,
                                input_shape=(224,224,3),
                                pooling='avg',classes=20,
                                weights='imagenet')

for (i,layer) in enumerate(vggmodel.layers):
    layer.trainable = False
    print((i, layer.name, layer.output_shape))

(0, 'input_2', [(None, 224, 224, 3)])
(1, 'block1_conv1', (None, 224, 224, 64))
(2, 'block1_conv2', (None, 224, 224, 64))
(3, 'block1_pool', (None, 112, 112, 64))
(4, 'block2_conv1', (None, 112, 112, 128))
(5, 'block2_conv2', (None, 112, 112, 128))
(6, 'block2_pool', (None, 56, 56, 128))
(7, 'block3_conv1', (None, 56, 56, 256))
(8, 'block3_conv2', (None, 56, 56, 256))
(9, 'block3_conv3', (None, 56, 56, 256))
(10, 'block3_pool', (None, 28, 28, 256))
(11, 'block4_conv1', (None, 28, 28, 512))
(12, 'block4_conv2', (None, 28, 28, 512))
(13, 'block4_conv3', (None, 28, 28, 512))
(14, 'block4_pool', (None, 14, 14, 512))
(15, 'block5_conv1', (None, 14, 14, 512))
(16, 'block5_conv2', (None, 14, 14, 512))
(17, 'block5_conv3', (None, 14, 14, 512))
(18, 'block5_pool', (None, 7, 7, 512))
(19, 'global_average_pooling2d', (None, 512))

```

New layers are connected to the model. The number of layers connected depends on the number of hidden layers missing. The equal number of new layers are added, followed by a prediction layer of units equalling to number of classes. If only a prediction layer is missing, another Dense Layer is added before adding a prediction layer.

```
In [10]: model = models.Sequential()

dense_layer_1 = Dense(32, activation='relu')
dense_layer_2 = Dense(32, activation='relu')
prediction_layer = Dense(20, activation='softmax')

model.add(vggmodel)
model.add(dense_layer_1)
model.add(dense_layer_2)
model.add(prediction_layer)

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 512)	14714688
dense (Dense)	(None, 32)	16416
dense_1 (Dense)	(None, 32)	1056
dense_2 (Dense)	(None, 20)	660

Total params: 14,732,820
 Trainable params: 18,132
 Non-trainable params: 14,714,688

The number of units in each newly added layers have been arbitrarily chosen to have 32. This is because more units in the newly connected layers were observed to lead to overfitting for the same number of classes. Hence, the number of units being added in any hidden layer requires trial and testing.

So, the pre-trained model now contains non-trainable hidden layers, and newly appended hidden layers of 32 units. The training involves having to train the newly connected layers only. The model is compiled by setting an optimizer, loss and metrics parameter values. The optimizer chosen is 'Adam' which performs optimization similar to Stochastic Gradient Descent. [42] 'categorical_crossentropy' is chosen as the loss, since the Transfer Learning models involve classifying across many different classes. 'accuracy' is chosen for the metrics.

```
In [11]: model.compile(  
        optimizer='adam',  
        loss='categorical_crossentropy',  
        metrics=['accuracy'],  
    )
```

```
In [12]: model.save("./models/action-class-20-model-vgg16.h5")
```

The models are saved in the storage and loaded. The loaded model from the storage is evaluated using the 'evaluate()' function. The evaluated validation accuracy and validation loss values are observed to be the same as that of the last epoch during training.

The models are trained with the training set of images and validated using validation set of images. The models are trained over 20 iterations, called *epochs*. Each epoch will involve having the entire training set to be used to train the model and set the neuron weights. 20 epochs are chosen to visualize the accuracy and loss values and to study the values. Tensorflow prints the *Training Accuracy, Training Loss, Validation Accuracy, Validation Loss, Training time for each epoch* and other values during the training.

```
In [13]: fit = model.fit(train_batches, epochs=20, validation_data=validation_  
Epoch 1/20
```

```

512/512 [=====] - 93s 172ms/step - loss:
2.3305 - accuracy: 0.3380 - val_loss: 1.6779 - val_accuracy: 0.4874
Epoch 2/20
512/512 [=====] - 93s 181ms/step - loss:
1.2268 - accuracy: 0.6440 - val_loss: 1.1752 - val_accuracy: 0.6236
Epoch 3/20
512/512 [=====] - 137s 267ms/step - loss:
0.8810 - accuracy: 0.7327 - val_loss: 1.0025 - val_accuracy: 0.6827
Epoch 4/20
512/512 [=====] - 148s 289ms/step - loss:
0.7065 - accuracy: 0.7903 - val_loss: 0.9571 - val_accuracy: 0.6992
Epoch 5/20
512/512 [=====] - 151s 293ms/step - loss:
0.5889 - accuracy: 0.8277 - val_loss: 0.8752 - val_accuracy: 0.7283
Epoch 6/20
512/512 [=====] - 150s 293ms/step - loss:
0.4970 - accuracy: 0.8537 - val_loss: 0.8522 - val_accuracy: 0.7346
Epoch 7/20
512/512 [=====] - 152s 297ms/step - loss:
0.4338 - accuracy: 0.8728 - val_loss: 0.8836 - val_accuracy: 0.7205
Epoch 8/20
512/512 [=====] - 152s 296ms/step - loss:
0.3743 - accuracy: 0.8877 - val_loss: 0.8541 - val_accuracy: 0.7157
Epoch 9/20
512/512 [=====] - 153s 298ms/step - loss:
0.3270 - accuracy: 0.9000 - val_loss: 0.7805 - val_accuracy: 0.7559
Epoch 10/20
512/512 [=====] - 152s 296ms/step - loss:
0.2936 - accuracy: 0.9150 - val_loss: 0.7711 - val_accuracy: 0.7638
Epoch 11/20
512/512 [=====] - 151s 294ms/step - loss:
0.2628 - accuracy: 0.9226 - val_loss: 0.7595 - val_accuracy: 0.7717
Epoch 12/20
512/512 [=====] - 151s 294ms/step - loss:
0.2314 - accuracy: 0.9345 - val_loss: 0.7738 - val_accuracy: 0.7661
Epoch 13/20
512/512 [=====] - 152s 296ms/step - loss:
0.2129 - accuracy: 0.9381 - val_loss: 0.7372 - val_accuracy: 0.7748
Epoch 14/20
512/512 [=====] - 152s 297ms/step - loss:
0.1936 - accuracy: 0.9472 - val_loss: 0.7900 - val_accuracy: 0.7591
Epoch 15/20
512/512 [=====] - 152s 296ms/step - loss:
0.1729 - accuracy: 0.9529 - val_loss: 0.8902 - val_accuracy: 0.7386
Epoch 16/20

```



```

512/512 [=====] - 152s 297ms/step - loss:
0.1566 - accuracy: 0.9576 - val_loss: 0.7687 - val_accuracy: 0.7709
Epoch 17/20
512/512 [=====] - 152s 296ms/step - loss:
0.1402 - accuracy: 0.9625 - val_loss: 0.8218 - val_accuracy: 0.7717
Epoch 18/20
512/512 [=====] - 152s 296ms/step - loss:
0.1322 - accuracy: 0.9646 - val_loss: 0.8053 - val_accuracy: 0.7677
Epoch 19/20
512/512 [=====] - 153s 297ms/step - loss:
0.1171 - accuracy: 0.9707 - val_loss: 0.7698 - val_accuracy: 0.7780
Epoch 20/20
512/512 [=====] - 153s 297ms/step - loss:
0.1068 - accuracy: 0.9721 - val_loss: 0.7963 - val_accuracy: 0.7772

```

With Tensorflow, model training history can be obtained. The values of Training Accuracy, Training Loss, Validation Accuracy and Validation Loss are obtained as lists. A line-graph has been plotted to observe the values obtained across 20 epochs. Also, the average of Validation Accuracy, Validation Loss and Training Time per epoch has been noted.

Evaluate

```
In [15]: model = models.load_model("./models/action-class-20-trained-vgg16.h5")
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 512)	14714688
dense (Dense)	(None, 32)	16416
dense_1 (Dense)	(None, 32)	1056
dense_2 (Dense)	(None, 20)	660

```

=====
Total params: 14,732,820
Trainable params: 18,132
Non-trainable params: 14,714,688

```

```
In [16]: model.evaluate(test_batches)
```

```

127/127 [=====] - 34s 238ms/step - loss:
0.7963 - accuracy: 0.7772

```

```
Out[16]: [0.7963224649429321, 0.7771653532981873]
```

The following modification has been done for the architectures of the pre-trained models Inception-V3, ResNet50, MobileNet and VGG-16:

3.2 InceptionV3 Transfer Learning

- **original architecture** - 'include_top=True'

```
(300, 'batch_normalization_85', (None, 8, 8, 320))
(301, 'activation_87', (None, 8, 8, 384))
(302, 'activation_88', (None, 8, 8, 384))
(303, 'activation_91', (None, 8, 8, 384))
(304, 'activation_92', (None, 8, 8, 384))
(305, 'batch_normalization_93', (None, 8, 8, 192))
(306, 'activation_85', (None, 8, 8, 320))
(307, 'mixed9_1', (None, 8, 8, 768))
(308, 'concatenate_1', (None, 8, 8, 768))
(309, 'activation_93', (None, 8, 8, 192))
(310, 'mixed10', (None, 8, 8, 2048))
(311, 'avg_pool', (None, 2048))
(312, 'predictions', (None, 1000))
```

- **without last few layers** - include_top=False

```
(300, 'batch_normalization_179', (None, 8, 8, 320))
(301, 'activation_181', (None, 8, 8, 384))
(302, 'activation_182', (None, 8, 8, 384))
(303, 'activation_185', (None, 8, 8, 384))
(304, 'activation_186', (None, 8, 8, 384))
(305, 'batch_normalization_187', (None, 8, 8, 192))
(306, 'activation_179', (None, 8, 8, 320))
(307, 'mixed9_1', (None, 8, 8, 768))
(308, 'concatenate_3', (None, 8, 8, 768))
(309, 'activation_187', (None, 8, 8, 192))
(310, 'mixed10', (None, 8, 8, 2048))
(311, 'global_average_pooling2d', (None, 2048))
```

- **Model Summary**

Model: "sequential"

Layer (type)	Output Shape	Param #
inception_v3 (Functional)	(None, 2048)	21802784
dense (Dense)	(None, 32)	65568
dense_1 (Dense)	(None, 5)	165

=====
Total params: 21,868,517
Trainable params: 65,733
Non-trainable params: 21,802,784
=====

Figure 6: Inception-V3 Transfer Learning Architecture

3.3 MobileNet Transfer Learning

- **original architecture** - 'include_top=True'

```
(70, 'conv_pw_11', (None, 14, 14, 512))
(71, 'conv_pw_11_bn', (None, 14, 14, 512))
(72, 'conv_pw_11_relu', (None, 14, 14, 512))
(73, 'conv_pad_12', (None, 15, 15, 512))
(74, 'conv_dw_12', (None, 7, 7, 512))
(75, 'conv_dw_12_bn', (None, 7, 7, 512))
(76, 'conv_dw_12_relu', (None, 7, 7, 512))
(77, 'conv_pw_12', (None, 7, 7, 1024))
(78, 'conv_pw_12_bn', (None, 7, 7, 1024))
(79, 'conv_pw_12_relu', (None, 7, 7, 1024))
(80, 'conv_dw_13', (None, 7, 7, 1024))
(81, 'conv_dw_13_bn', (None, 7, 7, 1024))
(82, 'conv_dw_13_relu', (None, 7, 7, 1024))
(83, 'conv_pw_13', (None, 7, 7, 1024))
(84, 'conv_pw_13_bn', (None, 7, 7, 1024))
(85, 'conv_pw_13_relu', (None, 7, 7, 1024))
(86, 'global_average_pooling2d', (None, 1, 1, 1024))
(87, 'dropout', (None, 1, 1, 1024))
(88, 'conv_preds', (None, 1, 1, 1000))
(89, 'reshape_2', (None, 1000))
(90, 'predictions', (None, 1000))
```

- **without last few layers** - include_top=False

```
(70, 'conv_pw_11', (None, 14, 14, 512))
```

```

(71, 'conv_pw_11_bn', (None, 14, 14, 512))
(72, 'conv_pw_11_relu', (None, 14, 14, 512))
(73, 'conv_pad_12', (None, 15, 15, 512))
(74, 'conv_dw_12', (None, 7, 7, 512))
(75, 'conv_dw_12_bn', (None, 7, 7, 512))
(76, 'conv_dw_12_relu', (None, 7, 7, 512))
(77, 'conv_pw_12', (None, 7, 7, 1024))
(78, 'conv_pw_12_bn', (None, 7, 7, 1024))
(79, 'conv_pw_12_relu', (None, 7, 7, 1024))
(80, 'conv_dw_13', (None, 7, 7, 1024))
(81, 'conv_dw_13_bn', (None, 7, 7, 1024))
(82, 'conv_dw_13_relu', (None, 7, 7, 1024))
(83, 'conv_pw_13', (None, 7, 7, 1024))
(84, 'conv_pw_13_bn', (None, 7, 7, 1024))
(85, 'conv_pw_13_relu', (None, 7, 7, 1024))
(86, 'global_average_pooling2d_1', (None, 1024))

```

- **Model Summary**

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
mobilenet_1.00_224 (Functional)	(None, 1024)	3228864
dense (Dense)	(None, 32)	32800
dense_2 (Dense)	(None, 32)	1056
dense_3 (Dense)	(None, 5)	165
=====		
Total params: 3,262,885		
Trainable params: 34,021		
Non-trainable params: 3,228,864		

Figure 7: MobileNet Transfer Learning Architecture

3.4 ResNet50 Transfer Learning

- **original architecture** - 'include_top=True'

```

(150, 'conv5_block1_3_conv', (None, 7, 7, 2048))
(151, 'conv5_block1_0_bn', (None, 7, 7, 2048))
(152, 'conv5_block1_3_bn', (None, 7, 7, 2048))
(153, 'conv5_block1_add', (None, 7, 7, 2048))

```

```

(154, 'conv5_block1_out', (None, 7, 7, 2048))
(155, 'conv5_block2_1_conv', (None, 7, 7, 512))
(156, 'conv5_block2_1_bn', (None, 7, 7, 512))
(157, 'conv5_block2_1_relu', (None, 7, 7, 512))
(158, 'conv5_block2_2_conv', (None, 7, 7, 512))
(159, 'conv5_block2_2_bn', (None, 7, 7, 512))
(160, 'conv5_block2_2_relu', (None, 7, 7, 512))
(161, 'conv5_block2_3_conv', (None, 7, 7, 2048))
(162, 'conv5_block2_3_bn', (None, 7, 7, 2048))
(163, 'conv5_block2_add', (None, 7, 7, 2048))
(164, 'conv5_block2_out', (None, 7, 7, 2048))
(165, 'conv5_block3_1_conv', (None, 7, 7, 512))
(166, 'conv5_block3_1_bn', (None, 7, 7, 512))
(167, 'conv5_block3_1_relu', (None, 7, 7, 512))
(168, 'conv5_block3_2_conv', (None, 7, 7, 512))
(169, 'conv5_block3_2_bn', (None, 7, 7, 512))
(170, 'conv5_block3_2_relu', (None, 7, 7, 512))
(171, 'conv5_block3_3_conv', (None, 7, 7, 2048))
(172, 'conv5_block3_3_bn', (None, 7, 7, 2048))
(173, 'conv5_block3_add', (None, 7, 7, 2048))
(174, 'conv5_block3_out', (None, 7, 7, 2048))
(175, 'avg_pool', (None, 2048))
(176, 'predictions', (None, 1000))

```

- **without last few layers** - include_top=False

```

(150, 'conv5_block1_3_conv', (None, 7, 7, 2048), True)
(151, 'conv5_block1_0_bn', (None, 7, 7, 2048), True)
(152, 'conv5_block1_3_bn', (None, 7, 7, 2048), True)
(153, 'conv5_block1_add', (None, 7, 7, 2048), True)
(154, 'conv5_block1_out', (None, 7, 7, 2048), True)
(155, 'conv5_block2_1_conv', (None, 7, 7, 512), True)
(156, 'conv5_block2_1_bn', (None, 7, 7, 512), True)
(157, 'conv5_block2_1_relu', (None, 7, 7, 512), True)
(158, 'conv5_block2_2_conv', (None, 7, 7, 512), True)
(159, 'conv5_block2_2_bn', (None, 7, 7, 512), True)
(160, 'conv5_block2_2_relu', (None, 7, 7, 512), True)
(161, 'conv5_block2_3_conv', (None, 7, 7, 2048), True)
(162, 'conv5_block2_3_bn', (None, 7, 7, 2048), True)
(163, 'conv5_block2_add', (None, 7, 7, 2048), True)
(164, 'conv5_block2_out', (None, 7, 7, 2048), True)
(165, 'conv5_block3_1_conv', (None, 7, 7, 512), True)
(166, 'conv5_block3_1_bn', (None, 7, 7, 512), True)
(167, 'conv5_block3_1_relu', (None, 7, 7, 512), True)
(168, 'conv5_block3_2_conv', (None, 7, 7, 512), True)
(169, 'conv5_block3_2_bn', (None, 7, 7, 512), True)

```

```
(170, 'conv5_block3_2_relu', (None, 7, 7, 512), True)
(171, 'conv5_block3_3_conv', (None, 7, 7, 2048), True)
(172, 'conv5_block3_3_bn', (None, 7, 7, 2048), True)
(173, 'conv5_block3_add', (None, 7, 7, 2048), True)
(174, 'conv5_block3_out', (None, 7, 7, 2048), True)
(175, 'avg_pool', (None, 2048), True)
```

- **Model Summary**

Model: "sequential"

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 2048)	23587712
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 32)	65568
dense_1 (Dense)	(None, 5)	165

Total params: 23,653,445
 Trainable params: 15,041,733
 Non-trainable params: 8,611,712

Figure 8: ResNet-50 Transfer Learning Architecture

3.5 VGG-16 Transfer Learning

- **original architecture** - 'include_top=True'

```
(0, 'input_1', [(None, 224, 224, 3)])
(1, 'block1_conv1', (None, 224, 224, 64))
(2, 'block1_conv2', (None, 224, 224, 64))
(3, 'block1_pool', (None, 112, 112, 64))
(4, 'block2_conv1', (None, 112, 112, 128))
(5, 'block2_conv2', (None, 112, 112, 128))
(6, 'block2_pool', (None, 56, 56, 128))
(7, 'block3_conv1', (None, 56, 56, 256))
(8, 'block3_conv2', (None, 56, 56, 256))
(9, 'block3_conv3', (None, 56, 56, 256))
(10, 'block3_pool', (None, 28, 28, 256))
(11, 'block4_conv1', (None, 28, 28, 512))
(12, 'block4_conv2', (None, 28, 28, 512))
(13, 'block4_conv3', (None, 28, 28, 512))
```

```

(14, 'block4_pool', (None, 14, 14, 512))
(15, 'block5_conv1', (None, 14, 14, 512))
(16, 'block5_conv2', (None, 14, 14, 512))
(17, 'block5_conv3', (None, 14, 14, 512))
(18, 'block5_pool', (None, 7, 7, 512))
(19, 'flatten', (None, 25088))
(20, 'fc1', (None, 4096))
(21, 'fc2', (None, 4096))
(22, 'predictions', (None, 1000))

```

- **without last few layers** - include_top=False

```

(0, 'input_2', [(None, 224, 224, 3)])
(1, 'block1_conv1', (None, 224, 224, 64))
(2, 'block1_conv2', (None, 224, 224, 64))
(3, 'block1_pool', (None, 112, 112, 64))
(4, 'block2_conv1', (None, 112, 112, 128))
(5, 'block2_conv2', (None, 112, 112, 128))
(6, 'block2_pool', (None, 56, 56, 128))
(7, 'block3_conv1', (None, 56, 56, 256))
(8, 'block3_conv2', (None, 56, 56, 256))
(9, 'block3_conv3', (None, 56, 56, 256))
(10, 'block3_pool', (None, 28, 28, 256))
(11, 'block4_conv1', (None, 28, 28, 512))
(12, 'block4_conv2', (None, 28, 28, 512))
(13, 'block4_conv3', (None, 28, 28, 512))
(14, 'block4_pool', (None, 14, 14, 512))
(15, 'block5_conv1', (None, 14, 14, 512))
(16, 'block5_conv2', (None, 14, 14, 512))
(17, 'block5_conv3', (None, 14, 14, 512))
(18, 'block5_pool', (None, 7, 7, 512))
(19, 'global_average_pooling2d', (None, 512))

```

- **Model Summary**

Model: "sequential"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 512)	14714688
dense (Dense)	(None, 32)	16416
dense_1 (Dense)	(None, 32)	1056
dense_2 (Dense)	(None, 5)	165
Total params: 14,732,325		
Trainable params: 17,637		
Non-trainable params: 14,714,688		

Figure 9: VGG-16 Transfer Learning Architecture

4 Evaluation

To test model efficiency, a comprehensive evaluation has been conducted. Specifically, the four models have been trained on different number of classifications – 5, 10, 15 and 20.

Their accuracies, loss values and training time have been recorded.

4.1 Inception-V3 Results

Classes	Average Validation Accuracy %	Average Validation Loss %	Avg Training Time per epoch (seconds)
5	95.846	12.343	22.5
10	91.863	24.732	51.65
15	93.56	23.13	81.05
20	88.823	40.978	109.55

Table 1: Inception-V3 Results for different classifications

For the different number of classifications for training Inception-V3 with Transfer Learning, the following results have been obtained.

The models have been trained for 20 epochs (iterations). For each iteration, the Training Accuracy, Training Loss, Validation Accuracy, Validation Loss and Training Time per epoch has been obtained and recorded. The above table shows the average of Accuracy and Loss in percentage and Average Training Time per epoch.

4.1.1 5 Action Classes

The Inception-V3 model has been trained using Transfer Learning for 5 action classifications from the dataset. The following values for accuracy, loss and training time have been displayed as output for each of the 20 epochs.

The following shows the last 10 epochs during training. All 20 epochs can be seen as output of the Python script in the Jupyter Notebook file:

```

Epoch 10/20
155/155 [=====] - 19s 123ms/step -
loss: 4.9945e-04 - accuracy: 1.0000 - val_loss: 0.1162 -
val_accuracy: 0.9609
Epoch 11/20
155/155 [=====] - 20s 127ms/step -
loss: 4.0790e-04 - accuracy: 1.0000 - val_loss: 0.1093 -
val_accuracy: 0.9635
Epoch 12/20
155/155 [=====] - 21s 134ms/step -
loss: 3.4676e-04 - accuracy: 1.0000 - val_loss: 0.1038 -
val_accuracy: 0.9714
Epoch 13/20
155/155 [=====] - 21s 138ms/step -
loss: 2.9887e-04 - accuracy: 1.0000 - val_loss: 0.1181 -
val_accuracy: 0.9609
Epoch 14/20
155/155 [=====] - 23s 148ms/step -
loss: 2.5659e-04 - accuracy: 1.0000 - val_loss: 0.1245 -
val_accuracy: 0.9583
Epoch 15/20
155/155 [=====] - 28s 183ms/step -
loss: 2.2497e-04 - accuracy: 1.0000 - val_loss: 0.1143 -
val_accuracy: 0.9661
Epoch 16/20
155/155 [=====] - 26s 168ms/step -
loss: 1.9509e-04 - accuracy: 1.0000 - val_loss: 0.1246 -
val_accuracy: 0.9609
Epoch 17/20
155/155 [=====] - 27s 173ms/step -
loss: 1.7427e-04 - accuracy: 1.0000 - val_loss: 0.1245 -
val_accuracy: 0.9583
Epoch 18/20
155/155 [=====] - 30s 196ms/step -
loss: 1.5146e-04 - accuracy: 1.0000 - val_loss: 0.1231 -
val_accuracy: 0.9609
Epoch 19/20
155/155 [=====] - 30s 192ms/step -
loss: 1.3396e-04 - accuracy: 1.0000 - val_loss: 0.1229 -
val_accuracy: 0.9609
Epoch 20/20
155/155 [=====] - 33s 210ms/step -
loss: 1.1848e-04 - accuracy: 1.0000 - val_loss: 0.1205 -
val_accuracy: 0.9635

```

The following plots depicts the Loss and Accuracies during training of the Machine Learning model Inception-V3 for 5 action classes. The loss reduced for each epoch close to 0. The Accuracy increased each epoch close to 1.

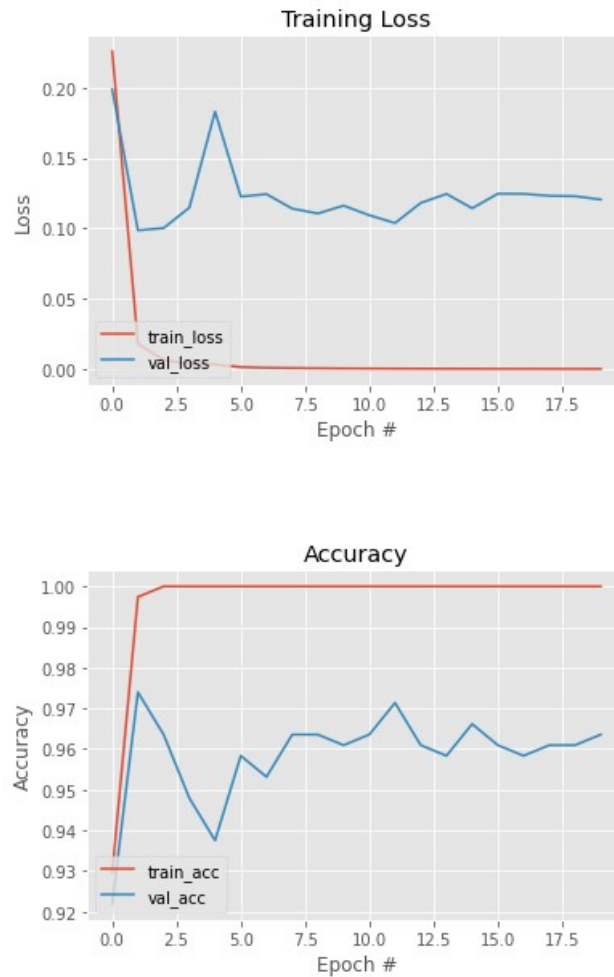


Figure 10: Loss and Accuracy observed in Inception-V3 for 5 Action Classes

```
In [18]: print("Avg Val Acc: " + str(sum(fit.history["val_accuracy"])/20*100))
print("Avg Val Loss: " + str(sum(fit.history["val_loss"])/20*100))
```

Avg Val Acc: 95.84635436534882
Avg Val Loss: 12.343161031603813

4.1.2 10 Action Classes

The Inception-V3 model has been trained using Transfer Learning for 10 action classifications from the dataset. The following values for accuracy, loss and training time have been displayed as output for each of the 20 epochs.

The following shows the last 10 epochs during training. All 20 epochs can be seen as output of the Python script in the Jupyter Notebook file:

```
Epoch 11/20
274/274 [=====] - 54s 198ms/step -
loss: 0.0010 - accuracy: 1.0000 - val_loss: 0.1929 -
val_accuracy: 0.9499
Epoch 12/20
274/274 [=====] - 55s 201ms/step -
loss: 8.3106e-04 - accuracy: 1.0000 - val_loss: 0.1784 -
val_accuracy: 0.9499
Epoch 13/20
274/274 [=====] - 67s 243ms/step -
loss: 6.6450e-04 - accuracy: 1.0000 - val_loss: 0.1716 -
val_accuracy: 0.9514
Epoch 14/20
274/274 [=====] - 67s 244ms/step -
loss: 5.5696e-04 - accuracy: 1.0000 - val_loss: 0.1921 -
val_accuracy: 0.9529
Epoch 15/20
274/274 [=====] - 69s 251ms/step -
loss: 4.6413e-04 - accuracy: 1.0000 - val_loss: 0.1878 -
val_accuracy: 0.9499
Epoch 16/20
274/274 [=====] - 63s 229ms/step -
loss: 3.8627e-04 - accuracy: 1.0000 - val_loss: 0.2123 -
val_accuracy: 0.9455
Epoch 17/20
274/274 [=====] - 61s 224ms/step -
loss: 3.3975e-04 - accuracy: 1.0000 - val_loss: 0.1893 -
val_accuracy: 0.9529
Epoch 18/20
274/274 [=====] - 73s 265ms/step -
loss: 2.7243e-04 - accuracy: 1.0000 - val_loss: 0.2038 -
val_accuracy: 0.9514
Epoch 19/20
274/274 [=====] - 68s 248ms/step -
loss: 2.2634e-04 - accuracy: 1.0000 - val_loss: 0.2160 -
val_accuracy: 0.9499
Epoch 20/20
```

```

274/274 [=====] - 75s 275ms/step -
loss: 2.1663e-04 - accuracy: 1.0000 - val_loss: 0.1968 -
val_accuracy: 0.9470

```

The following plots depicts the Loss and Accuracies during training of the Machine Learning model Inception-V3 for 10 action classes. The loss reduced for each epoch close to 0. The Accuracy increased each epoch close to 1.

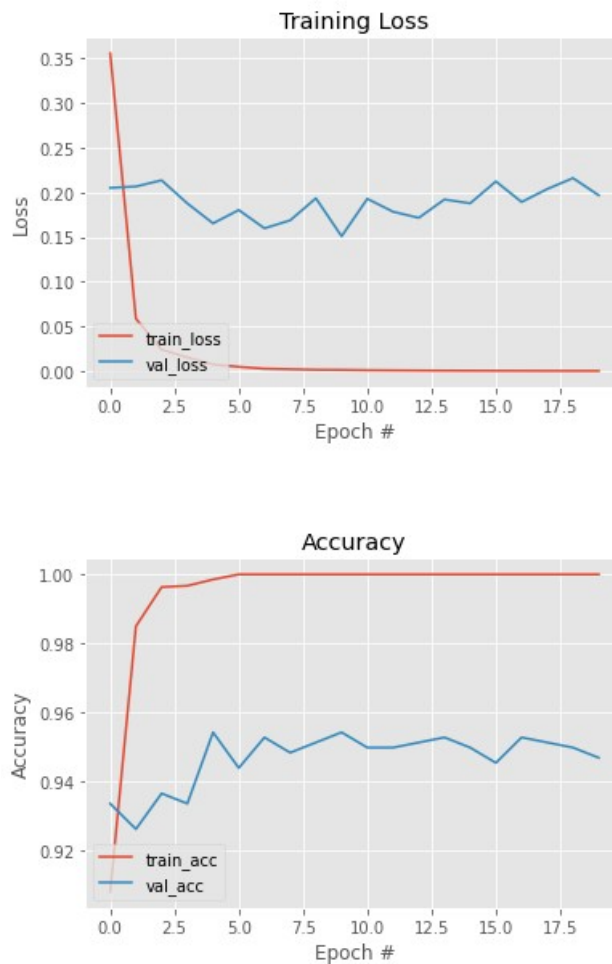


Figure 11: Loss and Accuracy observed in Inception-V3 for 10 Action Classes

4.1.3 15 Action Classes

The Inception-V3 model has been trained using Transfer Learning for 15 action classifications from the dataset. The following values for accuracy, loss and training time have been displayed as output for each of the 20 epochs.

The following shows the last 10 epochs during training. All 20 epochs can be seen as output of the Python script in the Jupyter Notebook file:

```
Epoch 11/20
377/377 [=====] - 89s 235ms/step -
loss: 0.0039 - accuracy: 1.0000 - val_loss: 0.1914 -
val_accuracy: 0.9465
Epoch 12/20
377/377 [=====] - 101s 268ms/step -
loss: 0.0029 - accuracy: 1.0000 - val_loss: 0.2185 -
val_accuracy: 0.9368
Epoch 13/20
377/377 [=====] - 98s 259ms/step -
loss: 0.0025 - accuracy: 1.0000 - val_loss: 0.2026 -
val_accuracy: 0.9411
Epoch 14/20
377/377 [=====] - 99s 261ms/step -
loss: 0.0021 - accuracy: 1.0000 - val_loss: 0.2029 -
val_accuracy: 0.9422
Epoch 15/20
377/377 [=====] - 85s 224ms/step -
loss: 0.0014 - accuracy: 1.0000 - val_loss: 0.2052 -
val_accuracy: 0.9475
Epoch 16/20
377/377 [=====] - 97s 257ms/step -
loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.2245 -
val_accuracy: 0.9465
Epoch 17/20
377/377 [=====] - 93s 246ms/step -
loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.2504 -
val_accuracy: 0.9325
Epoch 18/20
377/377 [=====] - 96s 255ms/step -
loss: 0.0382 - accuracy: 0.9875 - val_loss: 0.2758 -
val_accuracy: 0.9315
Epoch 19/20
377/377 [=====] - 97s 257ms/step -
loss: 0.0020 - accuracy: 1.0000 - val_loss: 0.3937 -
val_accuracy: 0.9111
Epoch 20/20
377/377 [=====] - 92s 243ms/step -
loss: 7.8810e-04 - accuracy: 1.0000 - val_loss: 0.2655 -
val_accuracy: 0.9379
```

The following plots depicts the Loss and Accuracies during training of the Machine Learning model Inception-V3 for 15 action classes. The loss reduced for each epoch close to 0. The Accuracy increased each epoch close to 1.

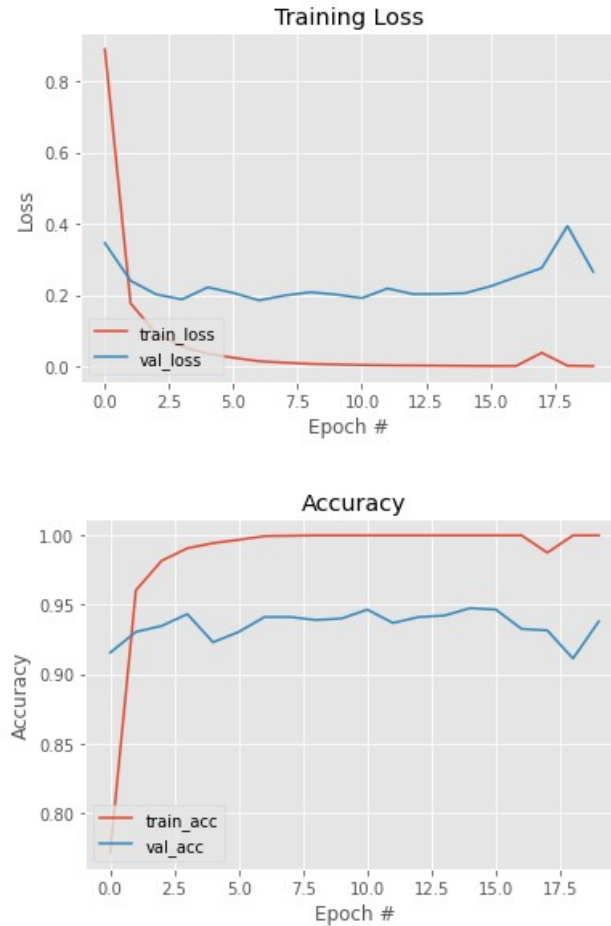


Figure 12: Loss and Accuracy observed in Inception-V3 for 15 Action Classes

```
In [18]: print("Avg Val Acc: " + str(sum(fit.history["val_accuracy"])/20))
print("Avg Val Loss: " + str(sum(fit.history["val_loss"])/20))
Avg Val Acc: 0.9355995684862137
Avg Val Loss: 0.23134735822677613
```

4.1.4 20 Action Classes

The Inception-V3 model has been trained using Transfer Learning for 20 action classifications from the dataset. The following values for accuracy, loss and training time have been displayed as output for each of the 20 epochs.

The following shows the last 10 epochs during training. All 20 epochs can be seen as output of the Python script in the Jupyter Notebook file:

```
Epoch 11/20
512/512 [=====] - 122s 238ms/step -
loss: 0.0028 - accuracy: 1.0000 - val_loss: 0.3508 -
val_accuracy: 0.9055
Epoch 12/20
512/512 [=====] - 130s 254ms/step -
loss: 0.0022 - accuracy: 1.0000 - val_loss: 0.3704 -
val_accuracy: 0.8992
Epoch 13/20
512/512 [=====] - 130s 254ms/step -
loss: 0.0017 - accuracy: 1.0000 - val_loss: 0.3765 -
val_accuracy: 0.8984
Epoch 14/20
512/512 [=====] - 134s 262ms/step -
loss: 0.0012 - accuracy: 1.0000 - val_loss: 0.4296 -
val_accuracy: 0.8921
Epoch 15/20
512/512 [=====] - 124s 243ms/step -
loss: 9.5946e-04 - accuracy: 1.0000 - val_loss: 0.3828 -
val_accuracy: 0.9063
Epoch 16/20
512/512 [=====] - 136s 266ms/step -
loss: 0.0290 - accuracy: 0.9920 - val_loss: 0.5124 -
val_accuracy: 0.8787
Epoch 17/20
512/512 [=====] - 134s 262ms/step -
loss: 0.0017 - accuracy: 1.0000 - val_loss: 0.4760 -
val_accuracy: 0.8866
Epoch 18/20
512/512 [=====] - 140s 273ms/step -
loss: 6.9804e-04 - accuracy: 1.0000 - val_loss: 0.4257 -
val_accuracy: 0.8945
Epoch 19/20
512/512 [=====] - 127s 247ms/step -
loss: 5.4271e-04 - accuracy: 1.0000 - val_loss: 0.4470 -
val_accuracy: 0.8921
Epoch 20/20
512/512 [=====] - 128s 249ms/step -
loss: 4.4166e-04 - accuracy: 1.0000 - val_loss: 0.4414 -
val_accuracy: 0.8984
```


The following plots depicts the Loss and Accuracies during training of the Machine Learning model Inception-V3 for 20 action classes. The loss reduced for each epoch close to 0. The Accuracy increased each epoch close to 1.

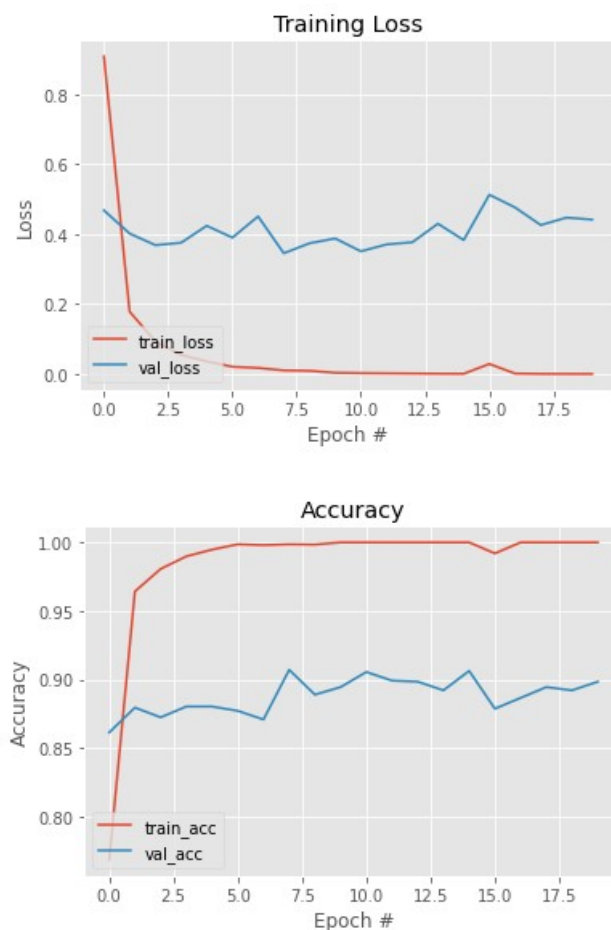


Figure 13: Loss and Accuracy observed in Inception-V3 for 20 Action Classes

```
In [18]: print("Avg Val Acc: " + str(sum(fit.history["val_accuracy"])/20))
          print("Avg Val Loss: " + str(sum(fit.history["val_loss"])/20))
```

```
Avg Val Acc: 0.8882283449172974
Avg Val Loss: 0.40977955162525176
```

4.2 MobileNet Results

Classes	Average Validation Accuracy %	Average Validation Loss %	Training Time per epoch (seconds)
5	95.898	13.412	5.3
10	92.18	26.7	11.1
15	91.825	29.818	17.5
20	86.842	47.973	25.42

Table 2: MobileNet results for different classifications

For the different number of classifications for training MobileNet with Transfer Learning, the following results have been obtained.

The models have been trained for 20 epochs (iterations). For each iteration, the Training Accuracy, Training Loss, Validation Accuracy, Validation Loss and Training Time per epoch has been obtained and recorded. The above table shows the average of Accuracy and Loss in percentage and Average Training Time per epoch.

4.2.1 5 Action Classes

The MobileNet model has been trained using Transfer Learning for 5 action classifications from the dataset. The following values for accuracy, loss and training time have been displayed as output for each of the 20 epochs.

The following shows the last 10 epochs during training. All 20 epochs can be seen as output of the Python script in the Jupyter Notebook file:

```
Epoch 11/20
155/155 [=====] - 5s 35ms/step -
loss: 1.9195e-04 - accuracy: 1.0000 - val_loss: 0.1093 -
val_accuracy: 0.9688
Epoch 12/20
155/155 [=====] - 5s 35ms/step -
loss: 1.6342e-04 - accuracy: 1.0000 - val_loss: 0.1073 -
val_accuracy: 0.9740
Epoch 13/20
155/155 [=====] - 5s 35ms/step -
loss: 1.4051e-04 - accuracy: 1.0000 - val_loss: 0.1098 -
val_accuracy: 0.9688
```

```

Epoch 14/20
155/155 [=====] - 5s 35ms/step -
loss: 1.2089e-04 - accuracy: 1.0000 - val_loss: 0.1085 -
val_accuracy: 0.9714
Epoch 15/20
155/155 [=====] - 5s 35ms/step -
loss: 1.0450e-04 - accuracy: 1.0000 - val_loss: 0.1089 -
val_accuracy: 0.9740
Epoch 16/20
155/155 [=====] - 5s 35ms/step -
loss: 9.1591e-05 - accuracy: 1.0000 - val_loss: 0.1101 -
val_accuracy: 0.9688
Epoch 17/20
155/155 [=====] - 5s 35ms/step -
loss: 8.0414e-05 - accuracy: 1.0000 - val_loss: 0.1109 -
val_accuracy: 0.9714
Epoch 18/20
155/155 [=====] - 6s 35ms/step -
loss: 7.1026e-05 - accuracy: 1.0000 - val_loss: 0.1096 -
val_accuracy: 0.9740
Epoch 19/20
155/155 [=====] - 6s 36ms/step -
loss: 6.2715e-05 - accuracy: 1.0000 - val_loss: 0.1106 -
val_accuracy: 0.9714
Epoch 20/20
155/155 [=====] - 6s 36ms/step -
loss: 5.5821e-05 - accuracy: 1.0000 - val_loss: 0.1119 -
val_accuracy: 0.9688

```

The following plots depicts the Loss and Accuracies during training of the Machine Learning model MobileNet for 5 action classes. The loss reduced for each epoch close to 0. The Accuracy increased each epoch close to 1.

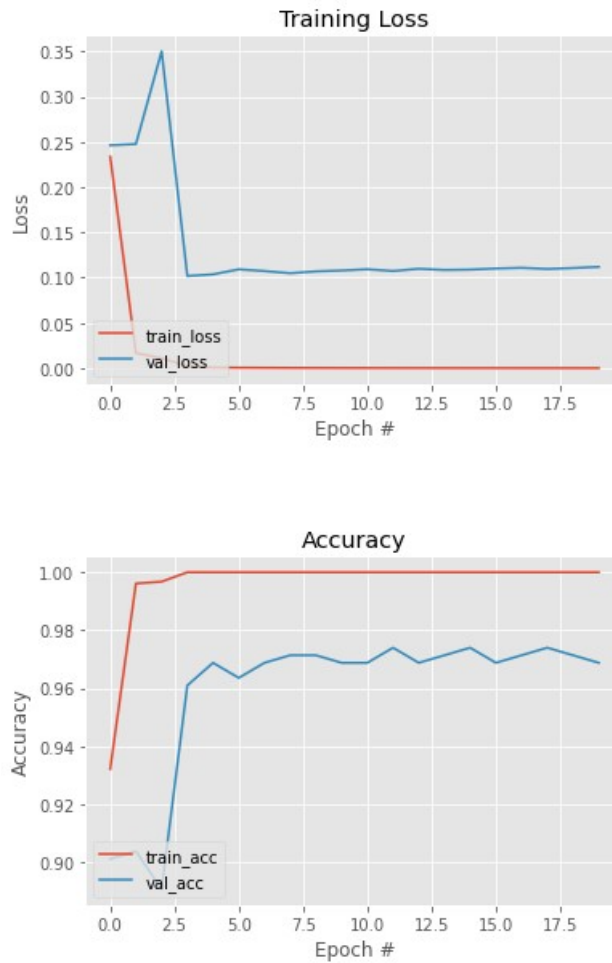


Figure 14: Loss and Accuracy observed in MobileNet for 5 Action Classes

```
In [18]: print("Avg Val Acc: " + str(sum(fit.history["val_accuracy"])/20*100))
print("Avg Val Loss: " + str(sum(fit.history["val_loss"])/20*100))
```

Avg Val Acc: 95.89843779802322
Avg Val Loss: 13.41185424476862

4.2.2 10 Action Classes

The MobileNet model has been trained using Transfer Learning for 10 action classifications from the dataset. The following values for accuracy, loss and training time have been displayed as output for each of the 20 epochs.

The following shows the last 10 epochs during training. All 20 epochs can be seen as output of the Python script in the Jupyter Notebook file:

```

Epoch 11/20
274/274 [=====] - 10s 37ms/step -
loss: 4.1977e-04 - accuracy: 1.0000 - val_loss: 0.2588 -
val_accuracy: 0.9264
Epoch 12/20
274/274 [=====] - 10s 37ms/step -
loss: 3.3263e-04 - accuracy: 1.0000 - val_loss: 0.2661 -
val_accuracy: 0.9278
Epoch 13/20
274/274 [=====] - 10s 38ms/step -
loss: 2.6883e-04 - accuracy: 1.0000 - val_loss: 0.2678 -
val_accuracy: 0.9308
Epoch 14/20
274/274 [=====] - 11s 39ms/step -
loss: 2.1617e-04 - accuracy: 1.0000 - val_loss: 0.2718 -
val_accuracy: 0.9308
Epoch 15/20
274/274 [=====] - 11s 39ms/step -
loss: 1.7567e-04 - accuracy: 1.0000 - val_loss: 0.2903 -
val_accuracy: 0.9249
Epoch 16/20
274/274 [=====] - 12s 43ms/step -
loss: 1.4695e-04 - accuracy: 1.0000 - val_loss: 0.2836 -
val_accuracy: 0.9264
Epoch 17/20
274/274 [=====] - 12s 45ms/step -
loss: 1.2083e-04 - accuracy: 1.0000 - val_loss: 0.2885 -
val_accuracy: 0.9234
Epoch 18/20
274/274 [=====] - 14s 49ms/step -
loss: 9.9981e-05 - accuracy: 1.0000 - val_loss: 0.2955 -
val_accuracy: 0.9219
Epoch 19/20
274/274 [=====] - 15s 56ms/step -
loss: 8.2963e-05 - accuracy: 1.0000 - val_loss: 0.2964 -
val_accuracy: 0.9234
Epoch 20/20
274/274 [=====] - 14s 50ms/step -
loss: 6.8844e-05 - accuracy: 1.0000 - val_loss: 0.2907 -
val_accuracy: 0.9308

```

The following plots depicts the Loss and Accuracies during training of the Machine Learning model MobileNet for 10 action classes. The loss reduced for each epoch close to 0. The Accuracy increased each epoch close to 1.

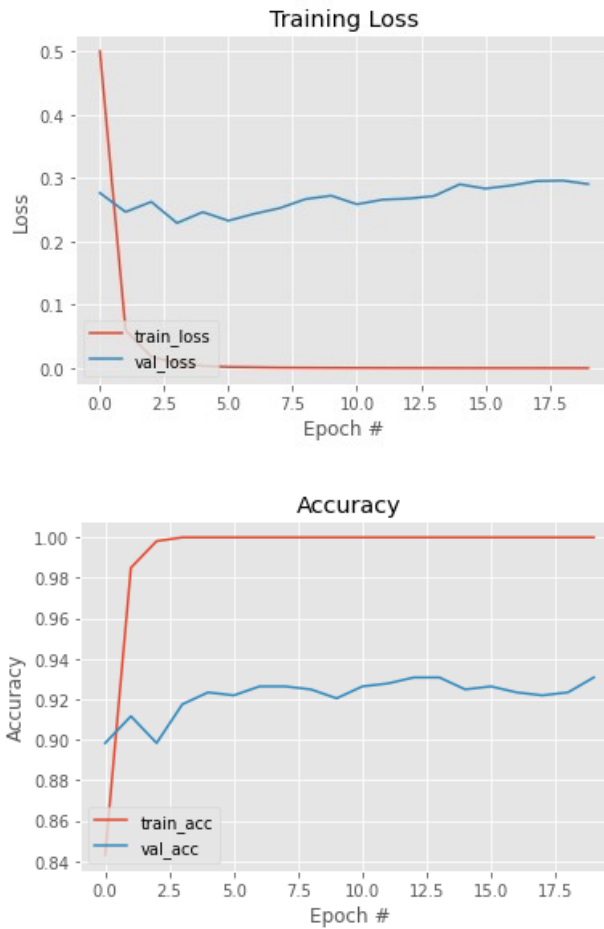


Figure 15: Loss and Accuracy observed in MobileNet for 10 Action Classes

```
In [18]: print("Avg Val Acc: " + str(sum(fit.history["val_accuracy"])/20*100))
print("Avg Val Loss: " + str(sum(fit.history["val_loss"])/20*100))
```

Avg Val Acc: 92.17967540025711
Avg Val Loss: 26.701502278447155

4.2.3 15 Action Classes

The MobileNet model has been trained using Transfer Learning for 15 action classifications from the dataset. The following values for accuracy, loss and training time have been displayed as output for each of the 20 epochs.

The following shows the last 10 epochs during training. All 20 epochs can be seen as output of the Python script in the Jupyter Notebook file:

```

Epoch 11/20
377/377 [=====] - 15s 40ms/step -
loss: 6.4720e-04 - accuracy: 1.0000 - val_loss: 0.2763 -
val_accuracy: 0.9229
Epoch 12/20
377/377 [=====] - 18s 47ms/step -
loss: 4.7841e-04 - accuracy: 1.0000 - val_loss: 0.2928 -
val_accuracy: 0.9208
Epoch 13/20
377/377 [=====] - 17s 44ms/step -
loss: 3.7209e-04 - accuracy: 1.0000 - val_loss: 0.2920 -
val_accuracy: 0.9218
Epoch 14/20
377/377 [=====] - 22s 57ms/step -
loss: 2.9139e-04 - accuracy: 1.0000 - val_loss: 0.2927 -
val_accuracy: 0.9197
Epoch 15/20
377/377 [=====] - 19s 51ms/step -
loss: 2.2899e-04 - accuracy: 1.0000 - val_loss: 0.2923 -
val_accuracy: 0.9229
Epoch 16/20
377/377 [=====] - 21s 55ms/step -
loss: 1.8166e-04 - accuracy: 1.0000 - val_loss: 0.3105 -
val_accuracy: 0.9218
Epoch 17/20
377/377 [=====] - 23s 61ms/step -
loss: 1.4695e-04 - accuracy: 1.0000 - val_loss: 0.2993 -
val_accuracy: 0.9251
Epoch 18/20
377/377 [=====] - 21s 55ms/step -
loss: 1.1475e-04 - accuracy: 1.0000 - val_loss: 0.3141 -
val_accuracy: 0.9240
Epoch 19/20
377/377 [=====] - 25s 65ms/step -
loss: 9.1354e-05 - accuracy: 1.0000 - val_loss: 0.3224 -
val_accuracy: 0.9218
Epoch 20/20
377/377 [=====] - 26s 68ms/step -
loss: 7.3019e-05 - accuracy: 1.0000 - val_loss: 0.3254 -
val_accuracy: 0.9208

```

The following plots depicts the Loss and Accuracies during training of the Machine Learning model MobileNet for 15 action classes. The loss reduced for each epoch close to 0. The Accuracy increased each epoch close to 1.

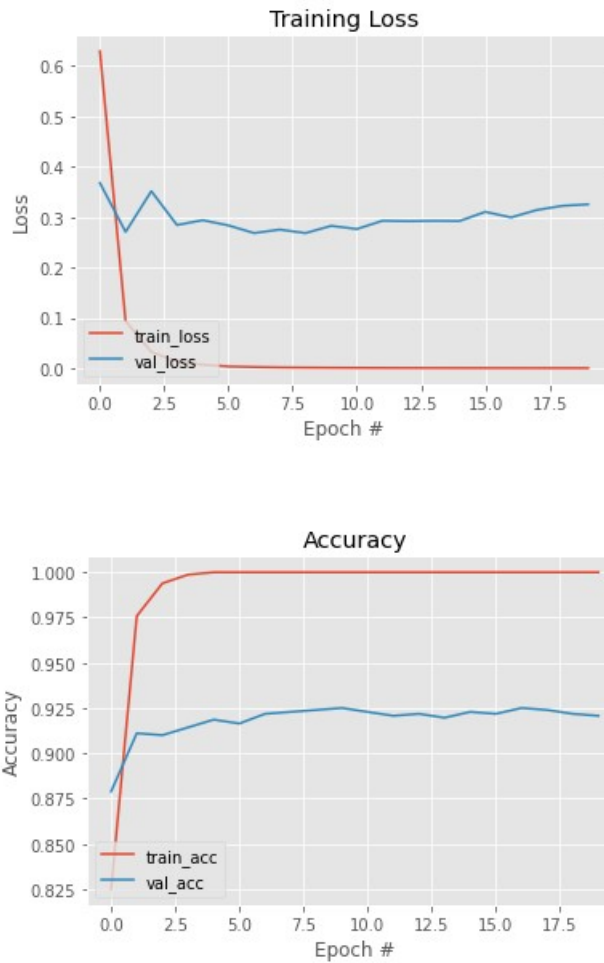


Figure 16: Loss and Accuracy observed in MobileNet for 15 Action Classes

```
In [18]: print("Avg Val Acc: " + str(sum(fit.history["val_accuracy"])/20))
          print("Avg Val Loss: " + str(sum(fit.history["val_loss"])/20))
```

```
Avg Val Acc: 0.9182548135519027
Avg Val Loss: 0.29817948341369627
```

4.2.4 20 Action Classes

The MobileNet model has been trained using Transfer Learning for 20 action classifications from the dataset. The following values for accuracy, loss and training time have been displayed as output for each of the 20 epochs.

The following shows the last 10 epochs during training. All 20 epochs can be seen as output of the Python script in the Jupyter Notebook file:


```

Epoch 11/20
512/512 [=====] - 28s 54ms/step -
loss: 9.0547e-04 - accuracy: 1.0000 - val_loss: 0.4368 -
val_accuracy: 0.8866
Epoch 12/20
512/512 [=====] - 28s 55ms/step -
loss: 6.5826e-04 - accuracy: 1.0000 - val_loss: 0.4334 -
val_accuracy: 0.8866
Epoch 13/20
512/512 [=====] - 30s 59ms/step -
loss: 5.1006e-04 - accuracy: 1.0000 - val_loss: 0.4446 -
val_accuracy: 0.8874
Epoch 14/20
512/512 [=====] - 29s 57ms/step -
loss: 3.8705e-04 - accuracy: 1.0000 - val_loss: 0.4590 -
val_accuracy: 0.8787
Epoch 15/20
512/512 [=====] - 31s 61ms/step -
loss: 2.9346e-04 - accuracy: 1.0000 - val_loss: 0.4573 -
val_accuracy: 0.8843
Epoch 16/20
512/512 [=====] - 31s 61ms/step -
loss: 2.1782e-04 - accuracy: 1.0000 - val_loss: 0.4613 -
val_accuracy: 0.8874
Epoch 17/20
512/512 [=====] - 36s 70ms/step -
loss: 1.6509e-04 - accuracy: 1.0000 - val_loss: 0.4718 -
val_accuracy: 0.8882
Epoch 18/20
512/512 [=====] - 32s 63ms/step -
loss: 1.2522e-04 - accuracy: 1.0000 - val_loss: 0.4602 -
val_accuracy: 0.8898
Epoch 19/20
512/512 [=====] - 32s 62ms/step -
loss: 9.2589e-05 - accuracy: 1.0000 - val_loss: 0.4771 -
val_accuracy: 0.8898
Epoch 20/20
512/512 [=====] - 32s 63ms/step -
loss: 7.0955e-05 - accuracy: 1.0000 - val_loss: 0.4960 -
val_accuracy: 0.8858

```

The following plots depicts the Loss and Accuracies during training of the Machine Learning model MobileNet for 20 action classes. The loss reduced for each epoch close to 0. The Accuracy increased each epoch close to 1.

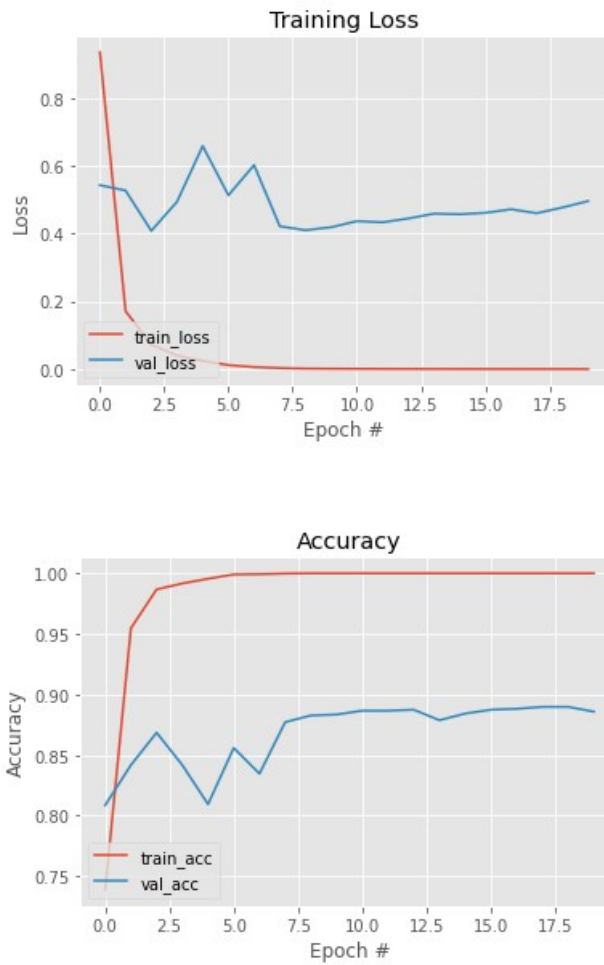


Figure 17: Loss and Accuracy observed in MobileNet for 20 Action Classes

```
In [19]: print("Avg Val Acc: " + str(sum(fit.history["val_accuracy"])/20*100))
print("Avg Val Loss: " + str(sum(fit.history["val_loss"])/20*100))
```

Avg Val Acc: 86.8425190448761
Avg Val Loss: 47.973386347293854

4.3 ResNet-50

Classes	Average Validation Accuracy %	Average Validation Loss %	Training Time per epoch (seconds)
5	95.208	14.995	26.7

10	92.29	23.943	58.4
15	92.671	29.605	79.8
20	90.48	34.083	100.6

Table 2: ResNet-50 Results for different classifications

For the different number of classifications for training ResNet-50 with Transfer Learning, the following results have been obtained.

The models have been trained for 20 epochs (iterations). For each iteration, the Training Accuracy, Training Loss, Validation Accuracy, Validation Loss and Training Time per epoch has been obtained and recorded. The above table shows the average of Accuracy and Loss in percentage and Average Training Time per epoch.

4.3.1 5 Action Classes

The ResNet-50 model has been trained using Transfer Learning for 5 action classifications from the dataset. The following values for accuracy, loss and training time have been displayed as output for each of the 20 epochs.

The following shows the last 10 epochs during training. All 20 epochs can be seen as output of the Python script in the Jupyter Notebook file:

```
Epoch 11/20
155/155 [=====] - 25s 161ms/step -
loss: 2.2908e-04 - accuracy: 1.0000 - val_loss: 0.1391 -
val_accuracy: 0.9557
Epoch 12/20
155/155 [=====] - 28s 180ms/step -
loss: 1.9492e-04 - accuracy: 1.0000 - val_loss: 0.1461 -
val_accuracy: 0.9531
Epoch 13/20
155/155 [=====] - 32s 205ms/step -
loss: 1.6610e-04 - accuracy: 1.0000 - val_loss: 0.1377 -
val_accuracy: 0.9557
Epoch 14/20
155/155 [=====] - 35s 224ms/step -
loss: 1.4720e-04 - accuracy: 1.0000 - val_loss: 0.1473 -
val_accuracy: 0.9531
Epoch 15/20
```

```

155/155 [=====] - 36s 234ms/step -
loss: 1.2394e-04 - accuracy: 1.0000 - val_loss: 0.1478 -
val_accuracy: 0.9505
Epoch 16/20
155/155 [=====] - 33s 215ms/step -
loss: 1.0638e-04 - accuracy: 1.0000 - val_loss: 0.1477 -
val_accuracy: 0.9557
Epoch 17/20
155/155 [=====] - 29s 187ms/step -
loss: 9.5088e-05 - accuracy: 1.0000 - val_loss: 0.1615 -
val_accuracy: 0.9479
Epoch 18/20
155/155 [=====] - 33s 213ms/step -
loss: 8.3654e-05 - accuracy: 1.0000 - val_loss: 0.1437 -
val_accuracy: 0.9531
Epoch 19/20
155/155 [=====] - 38s 248ms/step -
loss: 7.3131e-05 - accuracy: 1.0000 - val_loss: 0.1588 -
val_accuracy: 0.9505
Epoch 20/20
155/155 [=====] - 36s 231ms/step -
loss: 6.5294e-05 - accuracy: 1.0000 - val_loss: 0.1650 -
val_accuracy: 0.9479

```

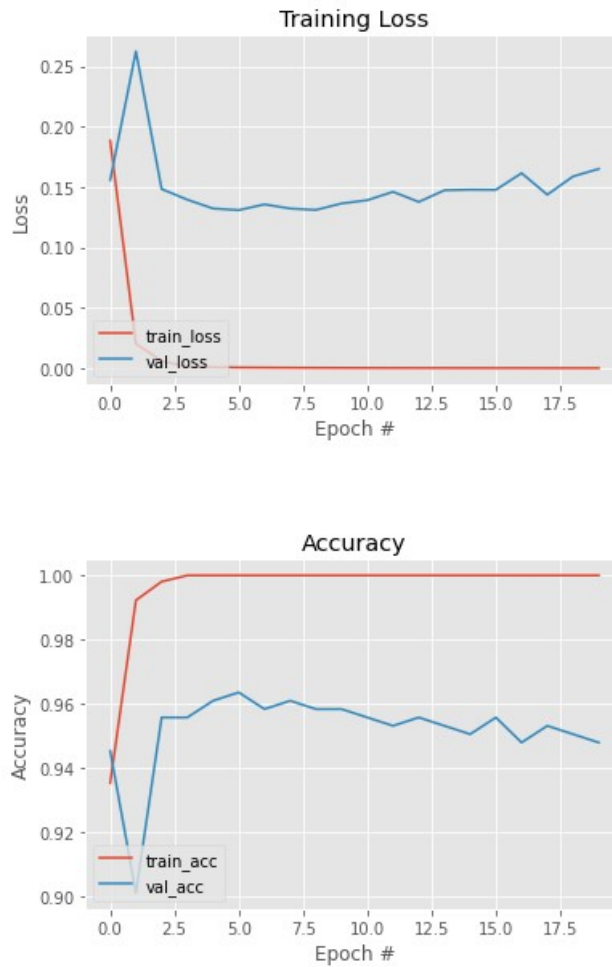


Figure 18: Loss and Accuracy observed in ResNet-50 for 5 Action Classes

```
In [18]: print("Avg Val Acc: " + str(sum(fit.history["val_accuracy"])/20*100))
print("Avg Val Loss: " + str(sum(fit.history["val_loss"])/20*100))

Avg Val Acc: 95.20833373069763
Avg Val Loss: 14.995239526033401
```

The following plots depicts the Loss and Accuracies during training of the Machine Learning model ResNet-50 for 5 action classes. The loss reduced for each epoch close to 0. The Accuracy increased each epoch close to 1.

4.3.2 10 Action Classes

The ResNet-50 model has been trained using Transfer Learning for 10 action classifications from the dataset. The following values for accuracy, loss and training time have been displayed as output for each of the 20 epochs.

The following shows the last 10 epochs during training. All 20 epochs can be seen as output of the Python script in the Jupyter Notebook file:

```
Epoch 11/20
274/274 [=====] - 64s 232ms/step -
loss: 4.0519e-04 - accuracy: 1.0000 - val_loss: 0.2359 -
val_accuracy: 0.9278
Epoch 12/20
274/274 [=====] - 62s 226ms/step -
loss: 3.2403e-04 - accuracy: 1.0000 - val_loss: 0.2428 -
val_accuracy: 0.9234
Epoch 13/20
274/274 [=====] - 65s 235ms/step -
loss: 2.6514e-04 - accuracy: 1.0000 - val_loss: 0.2398 -
val_accuracy: 0.9264
Epoch 14/20
274/274 [=====] - 50s 180ms/step -
loss: 2.1628e-04 - accuracy: 1.0000 - val_loss: 0.2397 -
val_accuracy: 0.9264
Epoch 15/20
274/274 [=====] - 68s 249ms/step -
loss: 1.7888e-04 - accuracy: 1.0000 - val_loss: 0.2455 -
val_accuracy: 0.9249
Epoch 16/20
274/274 [=====] - 63s 229ms/step -
loss: 1.5102e-04 - accuracy: 1.0000 - val_loss: 0.2409 -
val_accuracy: 0.9264
Epoch 17/20
274/274 [=====] - 65s 239ms/step -
loss: 1.2464e-04 - accuracy: 1.0000 - val_loss: 0.2484 -
val_accuracy: 0.9264
Epoch 18/20
274/274 [=====] - 64s 233ms/step -
loss: 1.0649e-04 - accuracy: 1.0000 - val_loss: 0.2579 -
val_accuracy: 0.9249
Epoch 19/20
274/274 [=====] - 68s 246ms/step -
loss: 8.6565e-05 - accuracy: 1.0000 - val_loss: 0.2656 -
val_accuracy: 0.9249
Epoch 20/20
```

```
274/274 [=====] - 65s 235ms/step -
loss: 7.3058e-05 - accuracy: 1.0000 - val_loss: 0.2732 -
val_accuracy: 0.9234
```

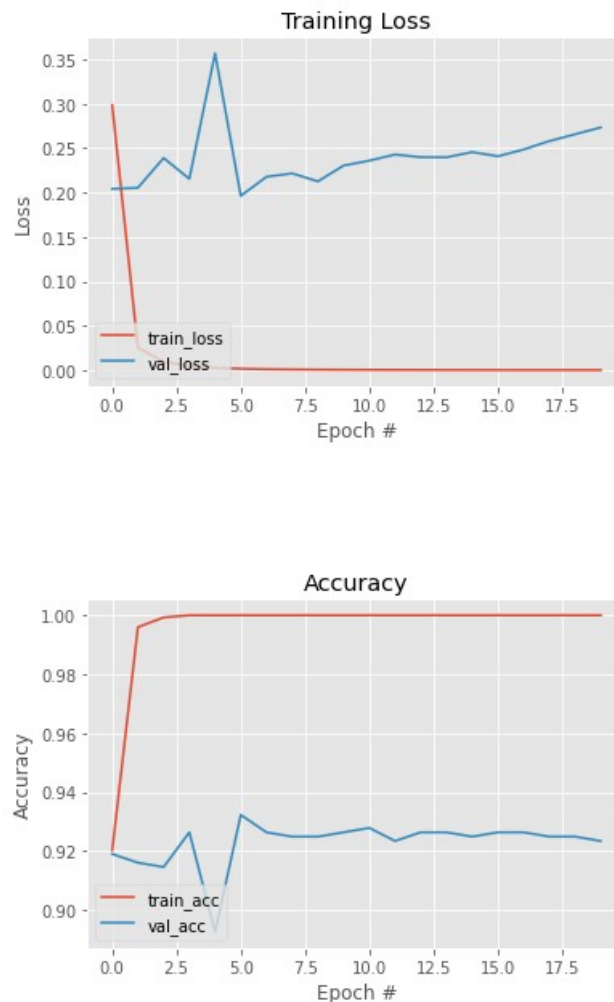


Figure 19: Loss and Accuracy observed in ResNet-50 for 10 Action Classes

```
In [18]: print("Avg Val Acc: " + str(sum(fit.history["val_accuracy"])/20*100))
          print("Avg Val Loss: " + str(sum(fit.history["val_loss"])/20*100))
          Avg Val Acc: 92.29013234376907
          Avg Val Loss: 23.943122550845146
```

The following plots depicts the Loss and Accuracies during training of the Machine Learning model ResNet-50 for 10 action classes. The loss reduced for each epoch close to 0. The Accuracy increased each epoch close to 1.

4.3.3 15 Action Classes

The ResNet-50 model has been trained using Transfer Learning for 15 action classifications from the dataset. The following values for accuracy, loss and training time have been displayed as output for each of the 20 epochs.

The following shows the last 10 epochs during training. All 20 epochs can be seen as output of the Python script in the Jupyter Notebook file:

```
Epoch 11/20
377/377 [=====] - 80s 213ms/step -
loss: 3.8374e-04 - accuracy: 1.0000 - val_loss: 0.3002 -
val_accuracy: 0.9261
Epoch 12/20
377/377 [=====] - 80s 211ms/step -
loss: 3.0551e-04 - accuracy: 1.0000 - val_loss: 0.2892 -
val_accuracy: 0.9315
Epoch 13/20
377/377 [=====] - 83s 220ms/step -
loss: 2.3718e-04 - accuracy: 1.0000 - val_loss: 0.2887 -
val_accuracy: 0.9293
Epoch 14/20
377/377 [=====] - 74s 195ms/step -
loss: 1.9346e-04 - accuracy: 1.0000 - val_loss: 0.3030 -
val_accuracy: 0.9315
Epoch 15/20
377/377 [=====] - 80s 214ms/step -
loss: 1.5707e-04 - accuracy: 1.0000 - val_loss: 0.3038 -
val_accuracy: 0.9315
Epoch 16/20
377/377 [=====] - 80s 212ms/step -
loss: 1.2265e-04 - accuracy: 1.0000 - val_loss: 0.3095 -
val_accuracy: 0.9293
Epoch 17/20
377/377 [=====] - 73s 194ms/step -
loss: 1.0048e-04 - accuracy: 1.0000 - val_loss: 0.2942 -
val_accuracy: 0.9315
Epoch 18/20
377/377 [=====] - 74s 195ms/step -
loss: 8.0115e-05 - accuracy: 1.0000 - val_loss: 0.3129 -
val_accuracy: 0.9293
Epoch 19/20
```



```

377/377 [=====] - 62s 165ms/step -
loss: 6.5001e-05 - accuracy: 1.0000 - val_loss: 0.3129 -
val_accuracy: 0.9304
Epoch 20/20
377/377 [=====] - 60s 158ms/step -
loss: 5.2101e-05 - accuracy: 1.0000 - val_loss: 0.3133 -
val_accuracy: 0.9336

```

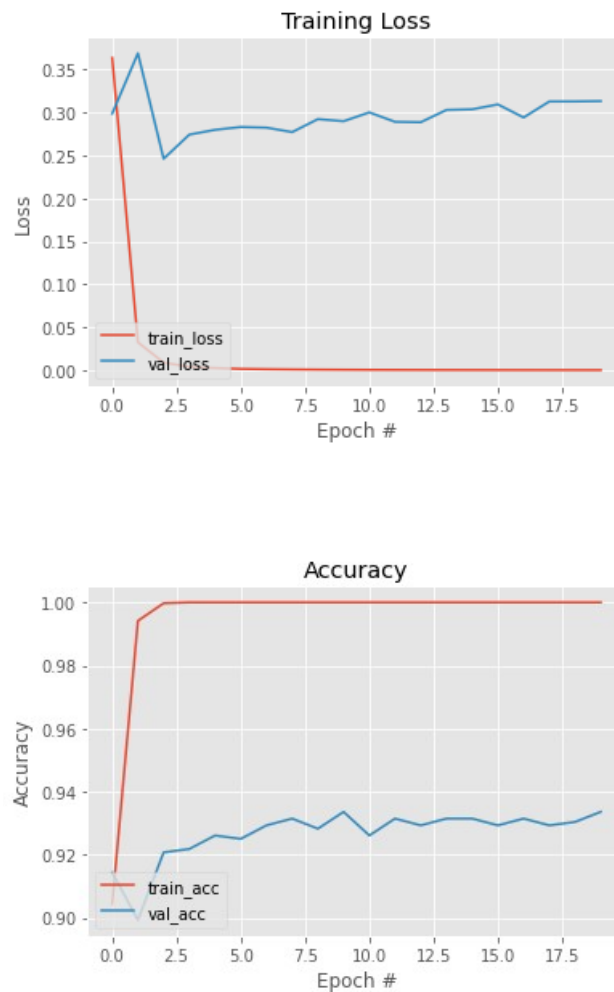


Figure 20: Loss and Accuracy observed in ResNet-50 for 15 Action Classes

```
In [18]: print("Avg Val Acc: " + str(sum(fit.history["val_accuracy"])/20*100))
print("Avg Val Loss: " + str(sum(fit.history["val_loss"])/20*100))

Avg Val Acc: 92.67130583524704
Avg Val Loss: 29.605583623051647
```

The following plots depicts the Loss and Accuracies during training of the Machine Learning model ResNet-50 for 15 action classes. The loss reduced for each epoch close to 0. The Accuracy increased each epoch close to 1.

4.3.4 20 Action Classes

The ResNet-50 model has been trained using Transfer Learning for 20 action classifications from the dataset. The following values for accuracy, loss and training time have been displayed as output for each of the 20 epochs.

The following shows the last 10 epochs during training. All 20 epochs can be seen as output of the Python script in the Jupyter Notebook file:

```
Epoch 11/20
512/512 [=====] - 108s 210ms/step -
loss: 3.9089e-04 - accuracy: 1.0000 - val_loss: 0.3322 -
val_accuracy: 0.9126
Epoch 12/20
512/512 [=====] - 107s 209ms/step -
loss: 3.0251e-04 - accuracy: 1.0000 - val_loss: 0.3215 -
val_accuracy: 0.9126
Epoch 13/20
512/512 [=====] - 112s 219ms/step -
loss: 2.1627e-04 - accuracy: 1.0000 - val_loss: 0.3522 -
val_accuracy: 0.9071
Epoch 14/20
512/512 [=====] - 128s 250ms/step -
loss: 1.6486e-04 - accuracy: 1.0000 - val_loss: 0.3346 -
val_accuracy: 0.9110
Epoch 15/20
512/512 [=====] - 120s 233ms/step -
loss: 1.2203e-04 - accuracy: 1.0000 - val_loss: 0.3404 -
val_accuracy: 0.9142
Epoch 16/20
512/512 [=====] - 125s 244ms/step -
loss: 9.3004e-05 - accuracy: 1.0000 - val_loss: 0.3649 -
val_accuracy: 0.9118
Epoch 17/20
```

```

512/512 [=====] - 120s 235ms/step -
loss: 7.0135e-05 - accuracy: 1.0000 - val_loss: 0.3543 -
val_accuracy: 0.9118
Epoch 18/20
512/512 [=====] - 124s 243ms/step -
loss: 5.1208e-05 - accuracy: 1.0000 - val_loss: 0.3681 -
val_accuracy: 0.9134
Epoch 19/20
512/512 [=====] - 110s 215ms/step -
loss: 3.8362e-05 - accuracy: 1.0000 - val_loss: 0.3779 -
val_accuracy: 0.9110
Epoch 20/20
512/512 [=====] - 113s 220ms/step -
loss: 2.8932e-05 - accuracy: 1.0000 - val_loss: 0.3803 -
val_accuracy: 0.9094

```

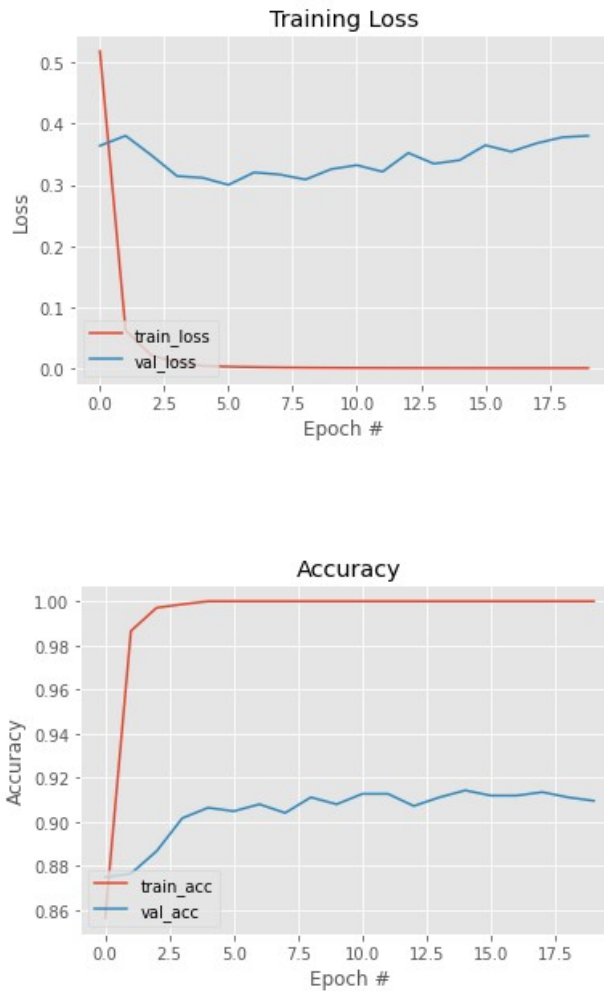


Figure 21: Loss and Accuracy observed in ResNet-50 for 20 Action Classes

```
In [18]: print("Avg Val Acc: " + str(sum(fit.history["val_accuracy"])/20*100))
print("Avg Val Loss: " + str(sum(fit.history["val_loss"])/20*100))
```

Avg Val Acc: 90.48031389713287
Avg Val Loss: 34.083402305841446

The following plots depicts the Loss and Accuracies during training of the Machine Learning model ResNet-50 for 20 action classes. The loss reduced for each epoch close to 0. The Accuracy increased each epoch close to 1.

4.4 VGG16 Results

Classes	Average Validation Accuracy %	Average Validation Loss %	Training Time per epoch (seconds)
5	85.234	44.293	40.65
10	81.068	55.707	75.8
15	76.938	70.244	109.5
20	72.937	88.71	140.2

Table 4: VGG-16 Results for different classifications

For the different number of classifications for training VGG-16 with Transfer Learning, the following results have been obtained.

The models have been trained for 20 epochs (iterations). For each iteration, the Training Accuracy, Training Loss, Validation Accuracy, Validation Loss and Training Time per epoch has been obtained and recorded. The above table shows the average of Accuracy and Loss in percentage and Average Training Time per epoch.

4.4.1 5 Action Classes

The VGG-16 model has been trained using Transfer Learning for 5 action classifications from the dataset. The following values for accuracy, loss and training time have been displayed as output for each of the 20 epochs.

The following shows the last 10 epochs during training. All 20 epochs can be seen as output of the Python script in the Jupyter Notebook file:

```
Epoch 11/20
155/155 [=====] - 46s 295ms/step -
loss: 0.0348 - accuracy: 0.9916 - val_loss: 0.4635 -
val_accuracy: 0.8568
Epoch 12/20
155/155 [=====] - 46s 293ms/step -
loss: 0.0340 - accuracy: 0.9929 - val_loss: 0.4530 -
val_accuracy: 0.8568
Epoch 13/20
155/155 [=====] - 46s 297ms/step -
loss: 0.0274 - accuracy: 0.9955 - val_loss: 0.4592 -
val_accuracy: 0.8281
Epoch 14/20
155/155 [=====] - 45s 292ms/step -
loss: 0.0230 - accuracy: 0.9961 - val_loss: 0.4194 -
val_accuracy: 0.8568
Epoch 15/20
155/155 [=====] - 47s 299ms/step -
loss: 0.0233 - accuracy: 0.9942 - val_loss: 0.4471 -
val_accuracy: 0.8646
Epoch 16/20
155/155 [=====] - 46s 297ms/step -
loss: 0.0170 - accuracy: 0.9981 - val_loss: 0.4513 -
val_accuracy: 0.8672
Epoch 17/20
155/155 [=====] - 46s 296ms/step -
loss: 0.0162 - accuracy: 0.9974 - val_loss: 0.3773 -
val_accuracy: 0.8828
Epoch 18/20
155/155 [=====] - 46s 294ms/step -
loss: 0.0127 - accuracy: 0.9981 - val_loss: 0.4037 -
val_accuracy: 0.8776
Epoch 19/20
155/155 [=====] - 47s 303ms/step -
loss: 0.0124 - accuracy: 0.9974 - val_loss: 0.4426 -
val_accuracy: 0.8620
Epoch 20/20
155/155 [=====] - 46s 294ms/step -
loss: 0.0117 - accuracy: 0.9974 - val_loss: 0.4558 -
val_accuracy: 0.8646
```

The following plots depicts the Loss and Accuracies during training of the Machine Learning model VGG-16 for 5 action classes. The loss reduced for each epoch close to 0. The Accuracy increased each epoch close to 1.

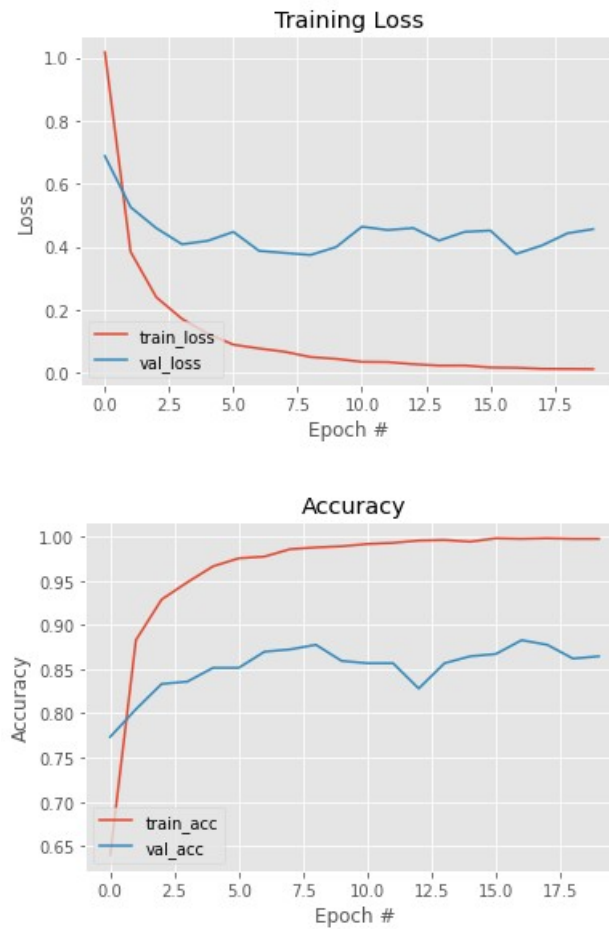


Figure 22: Loss and Accuracy observed in VGG-16 for 5 Action Classes

```
In [18]: print("Avg Val Acc: " + str(sum(fit.history["val_accuracy"])/20*100))
print("Avg Val Loss: " + str(sum(fit.history["val_loss"])/20*100))

Avg Val Acc: 85.23437470197678
Avg Val Loss: 44.29270029067993
```

4.4.2 10 Action Classes

The VGG-16 model has been trained using Transfer Learning for 10 action classifications from the dataset. The following values for accuracy, loss and training time have been displayed as output for each of the 20 epochs.

The following shows the last 10 epochs during training. All 20 epochs can be seen as output of the Python script in the Jupyter Notebook file:

```

Epoch 11/20
274/274 [=====] - 82s 297ms/step -
loss: 0.0907 - accuracy: 0.9755 - val_loss: 0.4386 -
val_accuracy: 0.8292
Epoch 12/20
274/274 [=====] - 81s 296ms/step -
loss: 0.0797 - accuracy: 0.9795 - val_loss: 0.6962 -
val_accuracy: 0.8159
Epoch 13/20
274/274 [=====] - 81s 295ms/step -
loss: 0.0699 - accuracy: 0.9817 - val_loss: 0.5364 -
val_accuracy: 0.8218
Epoch 14/20
274/274 [=====] - 81s 295ms/step -
loss: 0.0608 - accuracy: 0.9857 - val_loss: 0.4659 -
val_accuracy: 0.8336
Epoch 15/20
274/274 [=====] - 81s 296ms/step -
loss: 0.0543 - accuracy: 0.9872 - val_loss: 0.4637 -
val_accuracy: 0.8351
Epoch 16/20
274/274 [=====] - 81s 295ms/step -
loss: 0.0483 - accuracy: 0.9883 - val_loss: 0.4740 -
val_accuracy: 0.8306
Epoch 17/20
274/274 [=====] - 81s 294ms/step -
loss: 0.0418 - accuracy: 0.9898 - val_loss: 0.4922 -
val_accuracy: 0.8321
Epoch 18/20
274/274 [=====] - 81s 295ms/step -
loss: 0.0368 - accuracy: 0.9938 - val_loss: 0.5208 -
val_accuracy: 0.8247
Epoch 19/20
274/274 [=====] - 82s 297ms/step -
loss: 0.0358 - accuracy: 0.9901 - val_loss: 0.5735 -
val_accuracy: 0.8292
Epoch 20/20
274/274 [=====] - 82s 297ms/step -
loss: 0.0306 - accuracy: 0.9945 - val_loss: 0.5131 -
val_accuracy: 0.8306

```

The following plots depicts the Loss and Accuracies during training of the Machine Learning model VGG-16 for 10 action classes. The loss reduced for each epoch close to 0. The Accuracy increased each epoch close to 1.

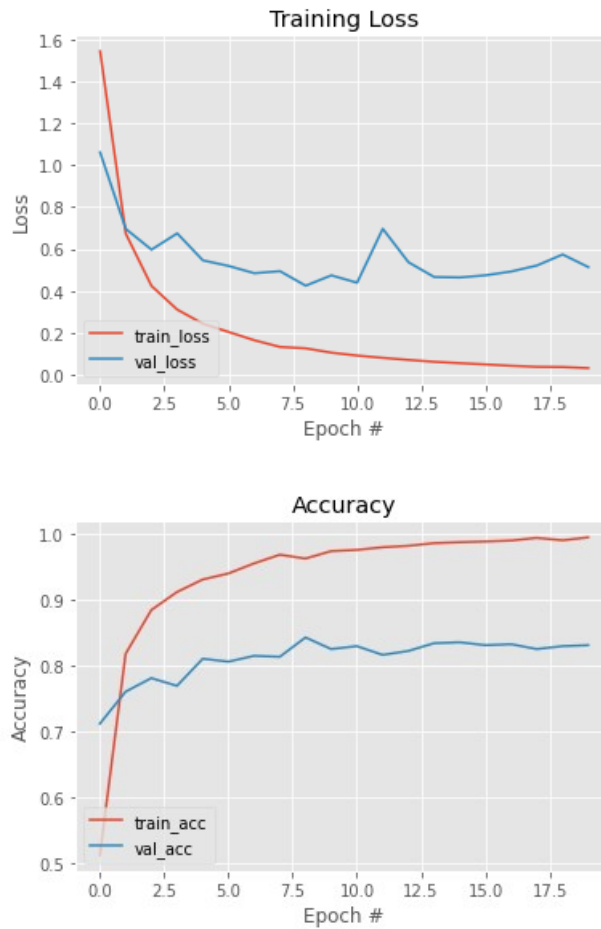


Figure 23: Loss and Accuracy observed in VGG-16 for 10 Action Classes

```
In [18]: print("Avg Val Acc: " + str(sum(fit.history["val_accuracy"])/20))
print("Avg Val Loss: " + str(sum(fit.history["val_loss"])/20))
Avg Val Acc: 0.8106774628162384
Avg Val Loss: 0.5570720851421356
```

4.4.3 15 Action Classes

The VGG-16 model has been trained using Transfer Learning for 15 action classifications from the dataset. The following values for accuracy, loss and training time have been displayed as output for each of the 20 epochs.

The following shows the last 10 epochs during training. All 20 epochs can be seen as output of the Python script in the Jupyter Notebook file:


```

Epoch 11/20
377/377 [=====] - 111s 294ms/step -
loss: 0.1578 - accuracy: 0.9586 - val_loss: 0.6983 -
val_accuracy: 0.7623
Epoch 12/20
377/377 [=====] - 112s 296ms/step -
loss: 0.1384 - accuracy: 0.9610 - val_loss: 0.8109 -
val_accuracy: 0.7281
Epoch 13/20
377/377 [=====] - 111s 293ms/step -
loss: 0.1313 - accuracy: 0.9631 - val_loss: 0.6609 -
val_accuracy: 0.7762
Epoch 14/20
377/377 [=====] - 114s 302ms/step -
loss: 0.1123 - accuracy: 0.9711 - val_loss: 0.6520 -
val_accuracy: 0.7827
Epoch 15/20
377/377 [=====] - 112s 297ms/step -
loss: 0.0956 - accuracy: 0.9774 - val_loss: 0.6752 -
val_accuracy: 0.7827
Epoch 16/20
377/377 [=====] - 113s 297ms/step -
loss: 0.0930 - accuracy: 0.9764 - val_loss: 0.7154 -
val_accuracy: 0.7762
Epoch 17/20
377/377 [=====] - 112s 297ms/step -
loss: 0.0788 - accuracy: 0.9814 - val_loss: 0.7033 -
val_accuracy: 0.7794
Epoch 18/20
377/377 [=====] - 112s 297ms/step -
loss: 0.0702 - accuracy: 0.9814 - val_loss: 0.6772 -
val_accuracy: 0.7848
Epoch 19/20
377/377 [=====] - 113s 298ms/step -
loss: 0.0643 - accuracy: 0.9851 - val_loss: 0.7302 -
val_accuracy: 0.7859
Epoch 20/20
377/377 [=====] - 112s 296ms/step -
loss: 0.0577 - accuracy: 0.9859 - val_loss: 0.7463 -
val_accuracy: 0.7837

```

The following plots depicts the Loss and Accuracies during training of the Machine Learning model VGG-16 for 10 action classes. The loss reduced for each epoch close to 0. The Accuracy increased each epoch close to 1.

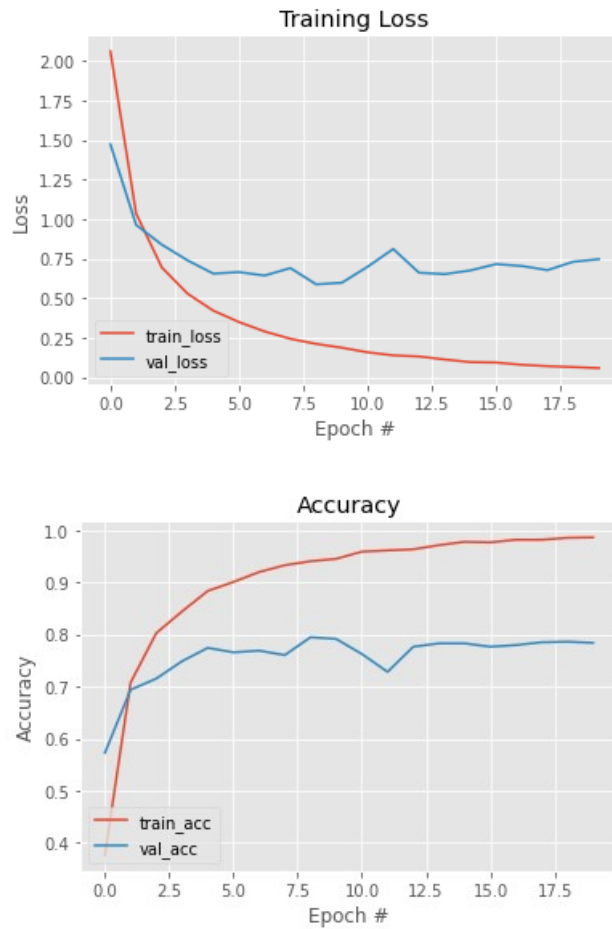


Figure 24: Loss and Accuracy observed in VGG-16 for 15 Action Classes

4.4.4 20 Action Classes

The VGG-16 model has been trained using Transfer Learning for 20 action classifications from the dataset. The following values for accuracy, loss and training time have been displayed as output for each of the 20 epochs.

The following shows the last 10 epochs during training. All 20 epochs can be seen as output of the Python script in the Jupyter Notebook file:

```
Epoch 11/20
512/512 [=====] - 151s 294ms/step -
loss: 0.2628 - accuracy: 0.9226 - val_loss: 0.7595 -
val_accuracy: 0.7717
Epoch 12/20
```

```

512/512 [=====] - 151s 294ms/step -
loss: 0.2314 - accuracy: 0.9345 - val_loss: 0.7738 -
val_accuracy: 0.7661
Epoch 13/20
512/512 [=====] - 152s 296ms/step -
loss: 0.2129 - accuracy: 0.9381 - val_loss: 0.7372 -
val_accuracy: 0.7748
Epoch 14/20
512/512 [=====] - 152s 297ms/step -
loss: 0.1936 - accuracy: 0.9472 - val_loss: 0.7900 -
val_accuracy: 0.7591
Epoch 15/20
512/512 [=====] - 152s 296ms/step -
loss: 0.1729 - accuracy: 0.9529 - val_loss: 0.8902 -
val_accuracy: 0.7386
Epoch 16/20
512/512 [=====] - 152s 297ms/step -
loss: 0.1566 - accuracy: 0.9576 - val_loss: 0.7687 -
val_accuracy: 0.7709
Epoch 17/20
512/512 [=====] - 152s 296ms/step -
loss: 0.1402 - accuracy: 0.9625 - val_loss: 0.8218 -
val_accuracy: 0.7717
Epoch 18/20
512/512 [=====] - 152s 296ms/step -
loss: 0.1322 - accuracy: 0.9646 - val_loss: 0.8053 -
val_accuracy: 0.7677
Epoch 19/20
512/512 [=====] - 153s 297ms/step -
loss: 0.1171 - accuracy: 0.9707 - val_loss: 0.7698 -
val_accuracy: 0.7780
Epoch 20/20
512/512 [=====] - 153s 297ms/step -
loss: 0.1068 - accuracy: 0.9721 - val_loss: 0.7963 -
val_accuracy: 0.7772

```

The following plots depicts the Loss and Accuracies during training of the Machine Learning model VGG-16 for 20 action classes. The loss reduced for each epoch close to 0. The Accuracy increased each epoch close to 1.

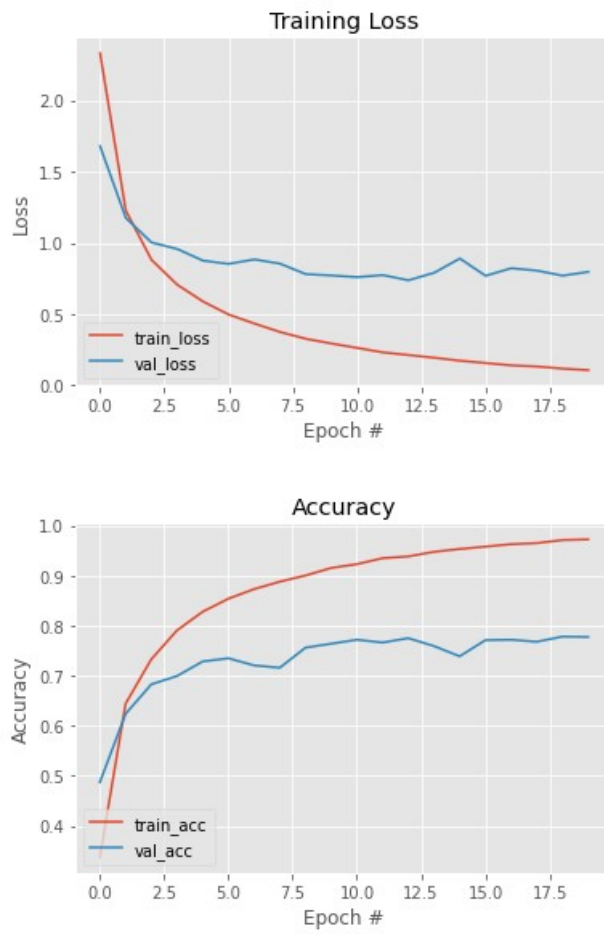


Figure 25: Loss and Accuracy observed in VGG-16 for 20 Action Classes

```
In [18]: print("Avg Val Acc: " + str(sum(fit.history["val_accuracy"])/20*100))
print("Avg Val Loss: " + str(sum(fit.history["val_loss"])/20*100))
```

```
Avg Val Acc: 72.93700769543648
Avg Val Loss: 88.70969265699387
```

5 Conclusion and Future Work

It has been observed that Accuracy of all models reduces as the number of classes involved in classification increases. The Loss, being inversely proportionate to Accuracy has been seen to gradually increase. The time taken to train the model for each epoch (iteration) increased with increase in number of classifications. Also, with each epoch involved in training the machine learning model, the accuracy increases close to 1 and loss reduces close to 0.

Model	Action Classes	Average Validation Accuracy %	Average Validation Loss %	Training Time per epoch (seconds)
Inception-V3	5	95.846	12.343	22.5
	10	91.863	24.732	51.65
	15	93.56	23.13	81.05
	20	88.823	40.978	109.55
MobileNet	5	95.898	13.412	5.3
	10	92.18	26.7	11.1
	15	91.825	29.818	17.5
	20	86.842	47.973	25.42
ResNet50	5	95.208	14.995	26.7
	10	92.29	23.943	58.4
	15	92.671	29.605	79.8
	20	90.48	34.083	100.6
VGG-16	5	85.234	44.293	40.65
	10	81.068	55.707	75.8
	15	76.938	70.244	109.5
	20	72.937	88.71	140.2

Table 5: Results for all models for different number of classifications

Loss is the difference between the outcome produced by the machine learning model and the intended outcome. Using the validation dataset, loss for each epoch

can be obtained. The parameters are the values that has to be tuned to get good model results by optimising a loss function. In the case of Neural Networks, the number of parameters is the measurement of the neural networks, comprising of layers in the network and the number of units in each layer. In this study, the number of parameters of each model is also compared to study the density and accuracies of the models.

Inception-V3 and MobileNet were in par with each other in terms of Accuracy and Loss. They both gave high accuracies and less loss. The training time for each epoch (iteration) for Inception-v3 was reasonable, ranging from 22 seconds to 110 seconds. Whereas, the training time for MobileNet for each epoch was very less – ranging between 5 seconds to 25 seconds.

Furthermore, Inception-V3 has 312 layers and 21,802,784 (21.8 million) parameters, making it a quite dense neural network architecture. In contrast, MobileNet has only 90 layers and 3,228,864 (3.2 million) parameters – It is not a very dense architecture. And yet, MobileNet seems to have given such high accuracy in its evaluation! So, MobileNet is observed to be very efficient and highly optimized – high accuracy and training for very little time and a sparse architecture. Inception-V3 is a great model to use for Transfer Learning for many more classes, hence, Inception-v3 can be used for predictions involving lots of classes.

ResNet50 implementation for Transfer Learning is slightly different. It involves having to use the in-built image pre-processing function for the training, validation and testing image datasets. It has seemed to have had the best performance in evaluation in this work! The accuracies were above 90% for all different number of classifications. It contains 23,587,712 (23.58 million) parameters in 176 layers. Lesser layers than Inception-V3 and almost as much dense in parameters in the network of neurons makes its training time lesser than Inception-V3. This model also performed better than Inception-V3 and MobileNet in terms of accuracies and loss.

VGG-16 is also a good model to implement for Image Classification and Transfer Learning. It is a simple model having only 16 layers, its accuracy was modest. It has only 16 layers, with more number of neurons in the layers. It has 14,714,688 (14.7 million) parameters in its architecture. The accuracy can be improved by making the last few layers more dense in number of neurons. More number of units (or neurons), like 512 or 1024 in number in each layer can help make the model more accurate. Also, more number of trainable layers within the architecture of VGG-16 can lead to higher accuracy.

Hence, based on the density of architecture with respect to number of parameters and layers, the accuracy values, loss values and training time per epoch, ResNet50 seemed to have overall the best architecture to implement for Transfer Learning – the Accuracy is the highest, Training Time is good, more number of pre-trained weights as parameters. MobileNet seemed to have the most efficient architecture – lesser number of parameters with pre-trained weights and yet very high accuracy, less training time. Followed by Inception-V3, more pre-trained weights as parameters and yet, slightly less accurate than MobileNet, more training time than ResNet50. VGG-16 performed poorly, but accuracy can be improved by having more trainable parameters.

Models trained using Transfer Learning tend to be overfit due to the training of only the last layers. The whole model can be trained during transfer learning, but a huge amount of data and computational resources are required. Datasets like Imagenet and CIFAR-100 are huge and ideal for deep learning techniques.

Accuracy can also be improved by adding more new Fully Connected layers to train. This requires a trial and error, to test the right number of layers and units to add to pre-trained models for a satisfactory accuracy value. By allowing more layers in pre-trained model to be trainable, more accuracy can be achieved. Both these methods will require more computational resources and time to train the models.

As part of the future work of this research and proposed work, the same model architectures can be trained for more Action Classes in UCF101 Dataset. Training for 50, 70, 90 and all 101 action classes in the dataset can be attempted. The models' architectures can be optimized. A trial and error for how many new Fully Connected Layers and how many Trainable Layers in pre-trained model can be performed. Although, it would be preferable to only use new Fully Connected Layers.

Further to the above mentioned future work, AlexNet as a pre-trained model can be used. A study can be made with the AlexNet architecture with the weights being set after training with 'Imagenet' or 'CIFAR-100' datasets.

Prediction can be attempted for the models on the chosen number of classifications. In real-time, frames can be extracted from Live Videos for the models to predict from. The models with well tried-and-tested Transfer Learning Architectures will be able to classify the actions performed by actors in the scene in real-time with high accuracy.

6 Self-Examination

6.1 Strengths

6.1.1 Breaking down big task into smaller chunks

I like to break down a big monolithic task or project into smaller chunks of processes. This helps me define attainable goals and work on the project task by task into a big whole. It helped me gain clarity of thought through the project and also helped me prioritize the tasks and work on the individual tasks.

6.1.2 Organizing individual chunks of tasks

Breaking down one big task into several smaller tasks would not suffice to ease the process of working on the project and hence another important aspect would be to organize each of these sub-tasks. This allowed me to draw a road map of the entire project and follow a path in order to reach the end goal of completion of the project. I aimed at completing these tasks across a week. And so, over weeks of time on working on the project, the implementation and dissertation were being worked on in a systematic and organized manner.

6.1.3 Enjoying the process

Having picked this topic because of interest in the subject and to learn further on the subject, I couldn't help but enjoy the entire journey of the project. The knowledge I gained and the challenges I faced were overall a great experience. The problems and challenges faced were crucial for learning, while the completion of weekly tasks were small accomplishments. The overall process was a fun learning experience.

6.2 Weakness

6.2.1 Time Management

Managing time plays a crucial role in every aspect of a project. Despite dividing the entire project into smaller tasks, it was an overwhelming number of tasks to complete. A lot of time was involved in each of the task because I was inexperienced and had a lack of knowledge in certain domains and concepts. Understanding model architectures for Trasfer Learning was challenging at first but once I got hands-on experience and learned the right layers and density of neurons to add, it seemed a lot more simpler.

A lot of time was also spent in resolving errors that occurred during the model training process. However, this helped me learn through instant feedbacks when running into errors and further facilitated better time management.

6.2.2 Models testing

Model testing involved huge computational resources. I proceeded to utilise the existing GPU in my local machine. This led to more training time. Each of the four models had to be trained for four different number of classifications. Hence training sixteen models was a delicate operation that required time management, patience and focus.

7 Professional Issues

Below are the professional issues that can occur with the real-world use of Transfer Learning for Deep Learning Neural Network Models.

7.1 Usability

Any of the above models are deployed in the real world to recognize human action. For example, it could find use in classifying actions in a live footage of a general store, supermarket or bank to observe human actions. Although it may seem to replace a human at the security end in observing the actions of humans and alerting for suspicious actions, large computational resources are required to predict the action for each frame extracted from the video.

In a live video footage, frames must first be extracted. Soon after, the frames must be sent to the program that invokes the prediction function of the models. Then, the action is classified with some degree of accuracy.

The prediction step requires a very fast machine with a very good Graphical Processing Unit. It would be too costly to implement. Furthermore, prediction could take a few seconds for each image. This could lead to prediction of not all frames. Also, this machine running the prediction program throughout the working day would require to dissipate heat, and this machine would emit a lot of heat and needs to be maintained, possibly replaced, regularly.

7.2 Reliability

These models may give great accuracy for the actions. But it may not exactly be reliable in all cases. For example, the image dataset that has been trained on, and the use case or images in the real world would not meet the same standard metrics. They camera angles would be different, image slightly distorted, even inanimate mannequins in the real-world could be mis-interpreted as a human.

Thus, the image recognition is not the ideal use case for security purposes in any commercial building per se. A solution to this could be to implement action recognition using transfer learning for human skeletons instead of camera images. The same models with their existing architectures could be trained on a dataset containing action skeletons.

As part of Computer Vision, human skeletons can be extracted from a particular image instead of an entire pre-processed image. In this manner, the

action and the prediction data is clean in itself and ideal for any kind of action recognition, even in real-time live footage.

7.3 Privacy

The architectures of these models were so well designed and implemented, that they seemed to produce an impeccable accuracy. A well implemented Transfer Learning model can produce a very high accuracy. This prompts people to make use of these machine learning models in commercial and non-commercial uses to recognize actions by humans. These humans need not even be aware that they are being the subjects of such a use-case.

Hence, using of these action recognition models with high accuracy can compromise privacy of individuals. Even without their consent, they could be recorded. Their data could be stored. It brings forth malicious activities that could compromise the dignity and privacy of individuals.

Also, an attacker can launch an effective and efficient brute force attack that can create instances of input to trigger each target class with high confidence without any further knowledge outside of the pre-trained model [9]. So, security of implementing these models should be one of the top-most priorities.

7.4 Monopoly

With the existing technology, many Blue Chip companies have monopolised the concept of Image Recognition - from recognizing simple objects to translating board signs of another language to detecting the owner of a smart phone. These big companies have also gone out of the way of professional ethics for their business needs.

The advent of computer vision and image recognition with Neural Networks have prompted these corporations to also monetize this as a service, with lifetime or subscription-based business models. Improvements in accuracies and performance from different model architectures using Transfer Learning will only prompt more usage of these models, and incur profits for these corporations.

GPU sales could increase for utilizing these models, corporations could charge monthly subscription fees for using their server-based models, smaller business can sprout in order to provide image recognition based services (for example, in smart cars) with unreliable service and compromising law practice etc. More IT laws should take place to facilitate good practice and commerce in use of Machine learning models for image recognition.

8 References

- [1] Kutlugun, Esra and Eyupoglu, “Artificial Intelligence Methods Used in Computer Vision”, pp. 214-218, 5th International Conference on Computer Science and Engineering (UBMK), September 2020
- [2] A. A. Almisreb, N. Jamil and N. M. Din, "Utilizing AlexNet Deep Transfer Learning for Ear Recognition," 2018 Fourth International Conference on Information Retrieval and Knowledge Management (CAMP), pp. 1-5, 2018
- [3] Ruzhang Yang, “Classifying Hand Written Digits with Deep Learning”, *Intelligent Information Management*, Vol.10 No.2, March 2018
- [4] Faisal Dharma Adhinata, Nia Annisa Ferani Tanjung, Widi Widayat, Gracia Rizka Pasfica and Fadlan Raka Satura, “Comparative Study of VGG16 and MobileNetV2 for Masked Face Recognition”, *Jurnal Ilmiah Teknik Elektro Komputer dan Informatika (JITEKI)*, August 2021
- [5] Xin Jia, “Image recognition method based on deep learning”, *29th Chinese Control And Decision Conference (CCDC)*, pp. 4730-4735, July 2017
- [6] Mnih V, Kavukcuoglu K, Silver D, et al., “Human-level control through deep reinforcement learning”, *Journal of Nature*, Vol. 518, pp. 529-533, 2015
- [7] Yuying Ge, Ruimao Zhang, Lingyun Wu, Xiaogang Wang, Xiaoou Tang and Ping Luo, “DeepFashion2: A Versatile Benchmark for Detection, Pose Estimation, Segmentation and Re-Identification of Clothing Images”, *Computer Vision and Pattern Recognition (CVPR)*, 2019
- [8] Kafeng Wang, Xitong Gao, Yiren Zhao, Xingjian Li, Dejing Dou and Cheng-Zhong Xu, “Pay Attention to Features, Transfer Learn Faster CNNs”, *8th International Conference on Learning Representations (ICLR)*, 2020
- [9] Shahbaz Rezaei and Xin Liu, “A Target-Agnostic Attack on Deep Models: Exploiting Security Vulnerabilities of Transfer Learning”, *8th International Conference on Learning Representations (ICLR)*, September 2019
- [10] Parsa Saadatpanah, Ali Shafahi, Amin Ghiasi, Chen Zhu, Tom Goldstein, David Jacobs and Christoph Studer, “Adversarially robust transfer learning”, *International Conference on Learning Representations (ICLR)*, 2020
- [11] Michael Volpp, Lukas P. Fröhlich, Kirsten Fischer, Andreas Doerr, Stefan Falkner, Frank Hutter and Christian Daniel, “Meta-Learning Acquisition

- Functions for Transfer Learning in Bayesian Optimization”, *International Conference on Learning Representations (ICLR)*, 2020
- [12] Jianzhe Lin, Liang Zhao, Qi Wang, Rabab Ward and Z. Jane Wang, “DT-LET: Deep transfer learning by exploring where to transfer”, *Journal of Neurocomputing*, Vol. 390, pp. 99-107, January 2020
- [13] Fatima-Zohra Hamlili, Mustapha Khelifi , Ahmed Bouida and Moh13 Beladgham, “Transfer learning with Resnet-50 for detecting COVID-19 in chest X-ray images” *Indonesian Journal of Electrical Engineering and Computer Science*, Vol. 25, pp. 1458-1468, 2022
- [14] Madona B. Sahaai, G. R. Jothilakshmi, D. Ravikumar, Raghavendra Prasath and Saurav Singh, “ResNet-50 based deep neural network using transfer learning for brain tumor classification”, *AIP Conference Proceedings*, Vol. 2463, May 2022
- [15] A. S. B. Reddy and D. S. Juliet, “Transfer Learning with ResNet-50 for Malaria Cell-Image Classification”, *International Conference on Communication and Signal Processing (ICCSP)*, pp. 0945-0949, 2019
- [16] S. Jaju and M. Chandak, “A Transfer Learning Model Based on ResNet-50 for Flower Detection”, *International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, pp. 307-311, 2022
- [17] Zaid Taher Omer and Amel Hussein Abbas, “Image anomalies detection using transfer learning of ResNet-50 convolutional neural network”, *Indonesian Journal of Electrical Engineering and Computer Science (IJECS)*, Vol. 27, pp. 198-205, July 2022
- [18] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision”, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818-2826, 2016
- [19] Jignesh Chowdary, G., Pun, N.S., Sonbhadra, S.K. and Agarwal, S., “Face Mask Detection Using Transfer Learning of InceptionV3”, *Springer*, Vol. 12581, 2020
- [20] Vishwanath C. Burkapalli and Priyadarshini C. Patil, “Transfer Learning: Inception-V3 Based Custom Classification Approach for Food Images”, *Journal on Image And Video Processing (ICTACT)*, Vol. 11, Issue 01, August 2020
- [21] C. Wang et al., “Pulmonary Image Classification Based on Inception-v3 Transfer Learning Model”, in *IEEE Access*, Vol. 7, pp. 146533-146541, 2019
- [22] S. Degadwala, D. Vyas, H. Biswas, U. Chakraborty, and S. Saha, “Image Captioning Using Inception V3 Transfer Learning Model”, *6th International*

- Conference on Communication and Electronics Systems (ICCES)*, pp. 1103-1108, 2021
- [23] L. G. Falconí, M. Pérez and W. G. Aguilar, “Transfer Learning in Breast Mammogram Abnormalities Classification with Mobilenet and Nasnet”, *International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 109-114, 2019
- [24] H. Pan, Z. Pang, Y. Wang, Y. Wang, and L. Chen, “A New Image Recognition and Classification Method Combining Transfer Learning Algorithm and MobileNet Model for Welding Defects”, in *IEEE Access*, Vol. 8, pp. 119951-119960, 2020
- [25] X. Liu, Z. Jia, X. Hou, M. Fu, L. Ma and Q. Sun, “Real-time Marine Animal Images Classification by Embedded System Based on Mobilenet and Transfer Learning”, *OCEANS 2019 - Marseille*, pp. 1-5, 2019
- [26] A. Rajbongshi, T. Sarker, M. M. Ahamad and M. M. Rahman, “Rose Diseases Recognition using MobileNet,” *4th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, pp. 1-7, 2020
- [27] S. L. Rabano, M. K. Cabatuan, E. Sybingco, E. P. Dadios and E. J. Calilung, “Common Garbage Classification Using MobileNet”, *IEEE 10th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, pp. 1-4, 2018
- [28] M. Hon and N. M. Khan, “Towards Alzheimer's disease classification through transfer learning”, *IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 1166-1169, 2017
- [29] M. Shaha and M. Pawar, “Transfer Learning for Image Classification”, *Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pp. 656-660, 2018
- [30] T. Kaur and T. K. Gandhi, “Automated Brain Image Classification Based on VGG-16 and Transfer Learning”, *2019 International Conference on Information Technology (ICIT)*, pp. 94-98, 2019
- [31] A. Shamsi *et al.*, “An Uncertainty-Aware Transfer Learning-Based Framework for COVID-19 Diagnosis”, in *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 32, no. 4, pp. 1408-1417, April 2021

- [32] S. A. Prajapati, R. Nagaraj and S. Mitra, "Classification of dental diseases using CNN and transfer learning", *5th International Symposium on Computational and Business Intelligence (ISCBI)*, pp. 70-74, 2017
- [33] C. D. Gürkaynak and N. Arica, "A case study on transfer learning in convolutional neural networks", *26th Signal Processing and Communications Applications Conference (SIU)*, pp. 1-4, 2018
- [34] S. Nagae, S. Kawai and H. Nobuhara, "Transfer Learning Layer Selection Using Genetic Algorithm", *IEEE Congress on Evolutionary Computation (CEC)*, pp. 1-6, 2020
- [35] G. Vrbančič and V. Podgorelec, "Transfer Learning With Adaptive Fine-Tuning", in *IEEE Access*, vol. 8, pp. 196197-196211, 2020
- [36] K. Kaur, R. Dhir and K. Kumar, "Transfer Learning approach for analysis of epochs on Handwritten Digit Classification", *2nd International Conference on Secure Cyber Computing and Communications (ICSCCC)*, pp. 456-458, 2021
- [37] Y. Bo, Y. Lu and W. He, "Few-Shot Learning of Video Action Recognition Only Based on Video Contents", *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 584-593, 2020
- [38] Y. Luo and B. Yang, "Video motions classification based on CNN", *IEEE International Conference on Computer Science, Artificial Intelligence and Electronic Engineering (CSAIEE)*, pp. 335-338, 2021
- [39] J. Wang, S. Li, Z. Duan and Z. Yuan, "Rethinking Temporal-Related Sample for Human Action Recognition", *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2368-2372, 2020
- [40] J. Yan, F. Angelini and S. M. Naqvi, "Image Segmentation Based Privacy-Preserving Human Action Recognition for Anomaly Detection", *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8931-8935, 2020
- [41] H. T. Binh, M. T. Chau, A. Sugimoto and B. T. Duy, "Selecting active frames for action recognition with vote fusion method", *7th International Conference on Computer and Communication Engineering (ICCCE)*, pp. 161-166, 2018
- [42] Diederik P. Kingma and Jimmy Ba, "Adam: A Method for Stochastic Optimization", *3rd International Conference for Learning Representations*, San Diego, 2015

9 Appendix

9.1 Using the code

1. The zip file must be downloaded and extracted for the submission files.
2. The file structure of the submission files are of 3 types:
 - (a) Jupyter Notebook scripts for the 4 models for 4 action classes – 5, 10, 15, 20 classes, of the '.ipynb' format.
 - (b) 'models/' folder containing the saved machine models – trained and undtrained, of the format '.h5'.
 - (c) 'notebook-view/' folder containing PDF versions of the Jupyter Notebook files, for easier viewing of the output and execution of the scripts.

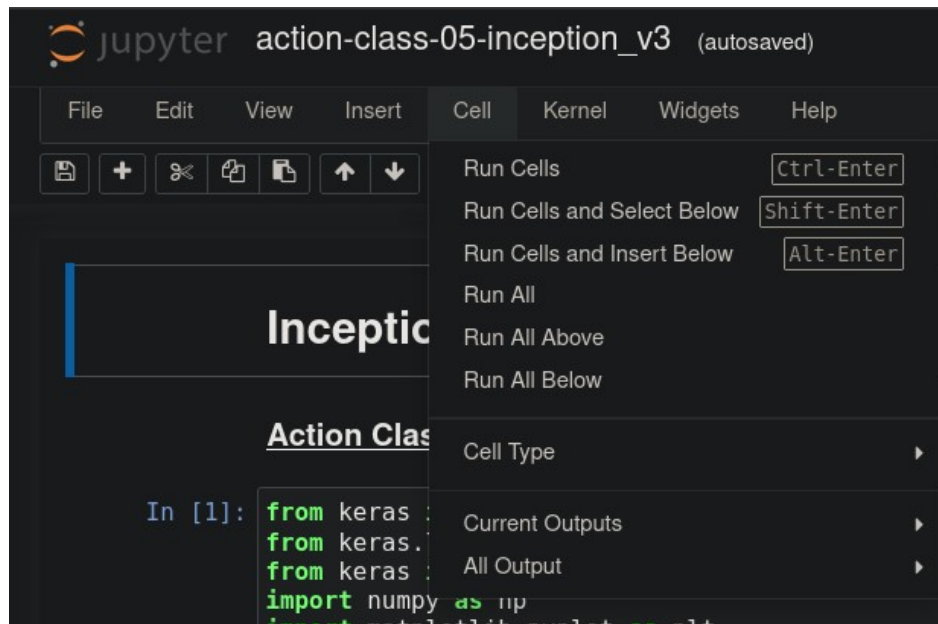
```
sankethbennur@pop-os:~/DISK2/uni/FINAL$ ls -a
.
..
action-class-05-inception_v3.ipynb
action-class-05-mobilenet.ipynb
action-class-05-resnet50.ipynb
action-class-05-vgg16.ipynb
action-class-10-inception_v3.ipynb
action-class-10-mobilenet.ipynb
action-class-10-resnet50.ipynb
action-class-10-vgg16.ipynb
action-class-15-inception_v3.ipynb
action-class-15-mobilenet.ipynb
action-class-15-resnet50.ipynb
action-class-15-vgg16.ipynb
action-class-20-inception_v3.ipynb
action-class-20-mobilenet.ipynb
action-class-20-resnet50.ipynb
action-class-20-vgg16.ipynb
dissertation.docx
extract-frames.ipynb
models
notebook-view
```



```
sankethbennur@pop-os:~/DISK2/uni/FINAL$ ls -a models/
.
..
action-class-05-inception_v3_model.h5
action-class-05-mobilenet.h5
action-class-05-resnet50.h5
action-class-05-trained-inception_v3_model.h5
action-class-05-trained-mobilenet.h5
action-class-05-trained-resnet50.h5
action-class-05-trained-vgg16.h5
action-class-05-vgg16.h5
action-class-10-model-inception_v3_model.h5
action-class-10-model-mobilenet.h5
action-class-10-model-vgg16.h5
action-class-10-resnet50.h5
action-class-10-trained-inception_v3_model.h5
action-class-10-trained-mobilenet.h5
action-class-10-trained-resnet50.h5
action-class-10-trained-vgg16.h5
action-class-15-model-inception_v3_model.h5
action-class-15-model-mobilenet.h5
action-class-15-model-vgg16.h5
action-class-15-resnet50.h5
action-class-15-trained-inception_v3_model.h5
action-class-15-trained-mobilenet.h5
action-class-15-trained-resnet50.h5
action-class-15-trained-vgg16.h5
action-class-20-model-inception_v3_model.h5
action-class-20-model-mobilenet.h5
action-class-20-model-vgg16.h5
action-class-20-resnet50.h5
action-class-20-trained-inception_v3_model.h5
action-class-20-trained-mobilenet.h5
action-class-20-trained-resnet50.h5
action-class-20-trained-vgg16.h5
```

```
sankethbennur@pop-os:~/DISK2/uni/FINAL$ ls -a notebook-view/
.
..
action-class-05-inception_v3.pdf
action-class-05-mobilenet.pdf
action-class-05-resnet50.pdf
action-class-05-vgg16.pdf
action-class-10-inception_v3.pdf
action-class-10-mobilenet.pdf
action-class-10-resnet50.pdf
'action-class-10-vgg16 .pdf'
action-class-15-inception_v3.pdf
action-class-15-mobilenet.pdf
action-class-15-resnet50.pdf
action-class-15-vgg16.pdf
action-class-20-inception_v3.pdf
action-class-20-mobilenet.pdf
action-class-20-resnet50.pdf
action-class-20-vgg16.pdf
```

3. Two other files exist:
 - (a) 'extract-frames.ipynb' – script to get the dataset required for the model training and evaluation.
 - (b) 'dissertation.pdf' – the dissertation file itself.
4. The UCF-101 dataset can be downloaded from:
`'https://www.crcv.ucf.edu/research/data-sets/ucf101/'`
 - (a) The zip folder downloaded contains 101 folders each containing videos of actions taken from YouTube.
 - (b) Folders of desired actions can be selected and extracted.
5. The Jupyter Notebook machine learning models' script files beginning with 'action-class-' can be opened through the Jupyter Notebook application.
 - (a) Jupyter Notebook must be first installed in the machine.
 - (b) Then, the application must be started to view the '.ipynb' files as a web server.
 - (c) On Linux based machines, the following terminal command will start the server:
`$ jupyter notebook`
6. The 'extract-frames.ipynb' file must be executed first, to obtain the dataset.
 - (a) The names of the action classes can be typed into the folder location string in the python code.
7. With the notebook opened in the server, the 'Run All' option can be selected in the 'Cell' option in the Menu Bar.



8. The notebooks for training the machine learning models can be run the same way.
9. The scripts in each cell will run till the very end of the file. Respective outputs of the cells can be viewed.

9.2 Code

9.2.1 Extract Frames from Videos

```
# Importing all necessary libraries
import cv2
import os

def get_frames(sourcedir, targetdir):
    if not os.path.exists(targetdir): os.mkdir(targetdir)

    for file in os.listdir(sourcedir):
        cap = cv2.VideoCapture(sourcedir+file)
        i = 0
        # a variable to set how many frames you want to skip
        frame_skip = 60
        # a variable to keep track of the frame to be saved
        frame_count = 0
        while cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                break
            if i > frame_skip - 1:
                frame_count += 1
                cv2.imwrite(targetdir + file +
str(frame_count*frame_skip)+'.jpg', frame)
                i = 0
                continue
            i += 1

        cap.release()
        cv2.destroyAllWindows()

# Extracting the frames from videos

get_frames("./UCF101-videos/ApplyLipstick/", "./frames/
ApplyLipstick/")
get_frames("./UCF101-videos/Archery/", "./frames/Archery/")
get_frames("./UCF101-videos/BabyCrawling/", "./frames/
BabyCrawling/")
get_frames("./UCF101-videos/Basketball/", "./frames/
Basketball/")
```

```

get_frames("./UCF101-videos/Biking/", "./frames/Biking/")
get_frames("./UCF101-videos/Diving/", "./frames/Diving/")
get_frames("./UCF101-videos/Fencing/", "./frames/Fencing/")
get_frames("./UCF101-videos/IceDancing/", "./frames/
IceDancing/")

get_frames("./UCF101-videos/Kayaking/", "./frames/Kayaking/")
get_frames("./UCF101-videos/MilitaryParade/", "./frames/
MilitaryParade/")
get_frames("./UCF101-videos/PizzaTossing/", "./frames/
PizzaTossing/")
get_frames("./UCF101-videos/PullUps/", "./frames/PullUps/")

get_frames("./UCF101-videos/ShavingBeard/", "./frames/
ShavingBeard/")
get_frames("./UCF101-videos/SkateBoarding/", "./frames/
SkateBoarding/")
get_frames("./UCF101-videos/SumoWrestling/", "./frames/
SumoWrestling/")
get_frames("./UCF101-videos/Surfing/", "./frames/Surfing/")

get_frames("./UCF101-videos/TennisSwing/", "./frames/
TennisSwing/")
get_frames("./UCF101-videos/Typing/", "./frames/Typing/")
get_frames("./UCF101-videos/WritingOnBoard/", "./frames/
WritingOnBoard/")
get_frames("./UCF101-videos/YoYo/", "./frames/YoYo/")

```

9.2.2 Train and Evaluate Machine Learning models

```

# Importing Libraries
from keras import models
from keras.layers import Dense, Flatten
from keras import backend as K
import numpy as np
import matplotlib.pyplot as plt
from keras.applications import vgg16

# Verifying GPU use
import tensorflow as tf
print("Num GPUs Available: ",
len(tf.config.list_physical_devices('GPU')))

from keras.preprocessing.image import ImageDataGenerator

```

```

dataset_path = "./frames/"
# will contain the categories in respective folders

# Data generators
train_datagen = ImageDataGenerator(rescale=1/255,
validation_split=0.2)

image_size = (224,224)
batch_size = 10

train_batches = train_datagen.flow_from_directory(
    dataset_path,
    target_size = image_size,
    batch_size = batch_size,
    class_mode = "categorical",
    subset = "training"
)

validation_batches = train_datagen.flow_from_directory(
    dataset_path,
    target_size = image_size,
    batch_size = batch_size,
    class_mode = "categorical",
    subset = "validation"
)

test_batches = train_datagen.flow_from_directory(
    dataset_path,
    target_size = image_size,
    batch_size = batch_size,
    class_mode = "categorical",
    subset = "validation"
)

# Depict the images in Dataset
from matplotlib import pyplot as plt

def plot_images(images_arr):
    fig, axes = plt.subplots(1,10)
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()

```

```

imgs, labels = train_batches[0]
plot_images(imgs)
print(labels[:10])

# Initialize the Model - without the top layers
vggmodel = vgg16.VGG16(include_top=False,
                        input_shape=(224,224,3),
                        pooling='avg',classes=20,
                        weights='imagenet')

for (i,layer) in enumerate(vggmodel.layers):
    layer.trainable = False
    print((i, layer.name, layer.output_shape))

# Add new layers
model = models.Sequential()

dense_layer_1 = Dense(32, activation='relu')
dense_layer_2 = Dense(32, activation='relu')
prediction_layer = Dense(20, activation='softmax')

model.add(vggmodel)
model.add(dense_layer_1)
model.add(dense_layer_2)
model.add(prediction_layer)

# Depict architecture of new model
model.summary()

# Compile the model ready to be trained
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy'],
)

# Train the Model
fit = model.fit(train_batches, epochs=20,
                validation_data=validation_batches)

# Save the model

```

```

model.save("./models/action-class-20-trained-vgg16.h5")

# Load the model + get summary of architecture
model = models.load_model("./models/action-class-20-trained-vgg16.h5")
model.summary()

# evaluate accuracy and loss of the trained model
model.evaluate(test_batches)

# Plot the model accuracy and loss - Training and Validation
plt.style.use("ggplot")
plt.figure()

plt.plot(np.arange(0, 20), fit.history["loss"],
label="train_loss")
plt.plot(np.arange(0, 20), fit.history["val_loss"],
label="val_loss")
plt.title("Training Loss")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend(loc="lower left")
plt.show()

plt.plot(np.arange(0, 20), fit.history["accuracy"],
label="train_acc")
plt.plot(np.arange(0, 20), fit.history["val_accuracy"],
label="val_acc")
plt.title("Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Accuracy")
plt.legend(loc="lower left")
plt.show()

```


9.2.3 Models stored in Hard-Drive

```
sankethbennur@pop-os:~/DISK2/uni$ ls -al models/
total 1666644
drwxrwxr-x 2 sankethbennur sankethbennur      4096 Aug 30 14:36 .
drwxrwxr-x 9 sankethbennur sankethbennur      4096 Aug 30 18:27 ..
-rw-rw-r-- 1 sankethbennur sankethbennur 87982976 Aug 25 22:18 action-class-05-inception_v3_model.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 13218728 Aug 25 22:35 action-class-05-mobilenet.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 98886000 Aug 30 14:10 action-class-05-resnet50.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 88524384 Aug 25 22:26 action-class-05-trained-inception_v3_model.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 13506296 Aug 25 22:37 action-class-05-trained-mobilenet.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 227211464 Aug 30 14:22 action-class-05-trained-resnet50.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 59146336 Aug 25 23:09 action-class-05-trained-vgg16.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 58988744 Aug 25 22:55 action-class-05-vgg16.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 87983736 Aug 25 14:18 action-class-10-model-inception_v3_model.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 13218912 Aug 25 17:12 action-class-10-model-mobilenet.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 58989768 Aug 25 15:22 action-class-10-model-vgg16.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 88527928 Aug 25 14:36 action-class-10-trained-inception_v3_model.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 13507600 Aug 25 17:16 action-class-10-trained-mobilenet.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 59148640 Aug 25 15:48 action-class-10-trained-vgg16.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 87984568 Aug 25 16:11 action-class-15-model-inception_v3_model.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 13219408 Aug 25 16:59 action-class-15-model-mobilenet.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 58989792 Aug 25 17:33 action-class-15-model-vgg16.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 88529400 Aug 25 16:38 action-class-15-trained-inception_v3_model.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 13508740 Aug 25 17:05 action-class-15-trained-mobilenet.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 59149088 Aug 25 18:08 action-class-15-trained-vgg16.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 87985080 Aug 25 18:43 action-class-20-model-inception_v3_model.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 13219856 Aug 25 19:35 action-class-20-model-mobilenet.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 58990304 Aug 25 20:01 action-class-20-model-vgg16.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 94944112 Aug 30 15:23 action-class-20-resnet50.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 88531064 Aug 25 19:20 action-class-20-trained-inception_v3_model.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 13512608 Aug 25 19:45 action-class-20-trained-mobilenet.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 59153472 Aug 25 20:49 action-class-20-trained-vgg16.h5
sankethbennur@pop-os:~/DISK2/uni$
```

```
sankethbennur@pop-os:~/DISK2/uni$ ls -al models/
total 1666644
drwxrwxr-x 2 sankethbennur sankethbennur      4096 Aug 30
14:36 .
drwxrwxr-x 9 sankethbennur sankethbennur      4096 Aug 30
18:27 ..
-rw-rw-r-- 1 sankethbennur sankethbennur 87982976 Aug 25
22:18 action-class-05-inception_v3_model.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 13218728 Aug 25
22:35 action-class-05-mobilenet.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 98886000 Aug 30
14:10 action-class-05-resnet50.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 88524384 Aug 25
22:26 action-class-05-trained-inception_v3_model.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 13506296 Aug 25
22:37 action-class-05-trained-mobilenet.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 227211464 Aug 30
14:22 action-class-05-trained-resnet50.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 59146336 Aug 25
23:09 action-class-05-trained-vgg16.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 58988744 Aug 25
22:55 action-class-05-vgg16.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 87983736 Aug 25
14:18 action-class-10-model-inception_v3_model.h5
```

```

-rw-rw-r-- 1 sankethbennur sankethbennur 13218912 Aug 25
17:12 action-class-10-model-mobilenet.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 58989768 Aug 25
15:22 action-class-10-model-vgg16.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 88527928 Aug 25
14:36 action-class-10-trained-inception_v3_model.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 13507600 Aug 25
17:16 action-class-10-trained-mobilenet.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 59148640 Aug 25
15:48 action-class-10-trained-vgg16.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 87984568 Aug 25
16:11 action-class-15-model-inception_v3_model.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 13219408 Aug 25
16:59 action-class-15-model-mobilenet.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 58989792 Aug 25
17:33 action-class-15-model-vgg16.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 88529400 Aug 25
16:38 action-class-15-trained-inception_v3_model.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 13508740 Aug 25
17:05 action-class-15-trained-mobilenet.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 59149088 Aug 25
18:08 action-class-15-trained-vgg16.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 87985080 Aug 25
18:43 action-class-20-model-inception_v3_model.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 13219856 Aug 25
19:35 action-class-20-model-mobilenet.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 58990304 Aug 25
20:01 action-class-20-model-vgg16.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 94944112 Aug 30
15:23 action-class-20-resnet50.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 88531064 Aug 25
19:20 action-class-20-trained-inception_v3_model.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 13512608 Aug 25
19:45 action-class-20-trained-mobilenet.h5
-rw-rw-r-- 1 sankethbennur sankethbennur 59153472 Aug 25
20:49 action-class-20-trained-vgg16.h5

```