Educational Administration: Theory and Practice
2024, 30(5), 2791 - 2798
ISSN: 2148-2403
https://kuey.net/

Research Article

A Comparative Analysis Using Transfer Learning

Ms. Swati Sharma[1*], Dr. Ruchi Sawhney[2], Dr. Santosh Kumar Singh[3], Dr. Varun Tiwari[4], Ms. Deepika Kirti[5], Dr. Vikas Rao Vadi[6]

[1*]Assistant Professor, Bosco Technical Training Society, Okhla Road, New Delhi, India
[2]Assistant Professor, Bosco Technical Training Society, Okhla Road, New Delhi, India
[3]Assistant Professor, Bosco Technical Training Society, Okhla Road, New Delhi, India
[4]Associate Professor, Don Bosco Institute of Technology, Okhla Road, New Delhi, India
[5]Assistant Professor, Bosco Technical Training Society, Okhla Road, New Delhi, India
[6]Professor & Director, Don Bosco Institute of Technology, Okhla Road, New Delhi, India

Citation: Ms. Swati Sharma, et.al (2024), A Comparative Analysis Using Transfer Learning, *Educational Administration: Theory and Practice*, *30*(5), 2791 - 2798
Doi: 10.53555/kuey.v30i5.3353

ARTICLE INFO

ABSTRACT

Through little adjustments, Transfer Learning enables us to leverage pre-trained models from other individuals. We will explain in this article how we may leverage pre-trained models to speed up our solutions. It is a well-liked deep learning strategy that has been effectively used in numerous areas and has recently drawn more attention from researchers. The growth in certain sectors is restricts when it becomes challenging to obtain a large-scale, well-annotated dataset because of the expense collecting the data. Transfer learning is a machine learning technique that involves applying a model created for one job to another task. In deep learning, transfer learning is gaining popularity because it allows deep neural networks to be trained with less data than if a model had to be created from the beginning. The assumption that the training and test sets should have the same distribution and be independent is lessened by transfer learning. The problem regarding insufficient training data is resolved through transfer learning. The examination of modern deep neural network-based transfer learning research and its applications is the main goal of this survey. We defined deep transfer learning, classified recent research papers using deep transfer learning approaches, and reviewed them.

Keywords: NAS Net Large, VGG19, InceptionResNetV2, Feature extraction, CNN

1. INTRODUCTION

The foundation of transfer learning is the adoption of features from one task and the "transfer" of the learnt information to another task. Domain adaptation is another phrase that's frequently used in the field of transfer learning. Domain adaptation is the process of adapting one or more source domains in order to facilitate knowledge transfer and enhance target learner performance [4]. This research paper's goal is to examine the possibility of using transfer learning to solve categorization problems more quickly, with fewer resources, and with less effort. In order to achieve better and faster results in the field of classification issues, we investigate three pretrained architectures (VGG19, NASNetLarge, and InceptionResNetV2) for transfer learning in this study. Since it is evident that we require ample data in order to train our complex convolution neural network, we combined our simple neural network with pretrained architectures (trained on a sizable ImageNet dataset) to create a complex model using the small dataset and minimizing the time and effort required to train a complex CNN.

PROPOSED WORK

RESEARCH OBJECTIVES:

- Develop a sophisticated CNN model and train it more quickly, accurately, and with less effort. Prevent overfitting due to dataset available.
- After adding a small architecture, we compared three pretrained designs to determine which is best, in context of accuracy and loss due to transfer learning.
- Potential Benefits:
- In the area of transfer learning, pretrained architectural performance is assessed.

- Avoid Overfitting: Hance the dataset is very small therefore the complex model can be engrossed to be overfitting.

IMPLEMENTATION

In this paper, we evaluate the different approaches of transfer learning to dataset of two species of the dog Chihuahua and Labrador for performance evaluation. In the paper, features are extracted through a pre-trained VGG19 model, but for classification, small CNN architecture is added. The following models were used in the experiment with 'ImageNet' weights.

- VGG19
- NASNetLarge
- InceptionResNetV2

ARCHITECTURES USED:

1. VGG19:

In order to categorize photos into 1000 object categories, Simonyan and Zisserman (2014) proposed the VGG19 convolutional neural network, which consists of 19 layers: 16 convolution layers and 3 fully connected layers. The ImageNet database, which has one million photos in 1000 categories, is used to train VGG19. Because each convolutional layer in the approach uses numerous 3 × 3 filters, it is a particularly common method for image categorization. Fig. 1 depicts the VGG19 architecture. This demonstrates that three more convolutional layers are employed for classification after 16 convolutional layers are used for feature extraction. A max-pooling layer comes after each of the five groups into which the feature extraction layers are divided [6]. Although VGG was developed by the Visual Geometry Group at Oxford University, hence the name VGG, it can be thought of as the successor to AlexNet, which debuted in 2012 and improved upon traditional convolutional neural networks. It borrows ideas from its predecessors, refines them, and employs deep convolutional neural layers to increase accuracy. To put it simply, VGG is a deep CNN that is used for image classification. The layers of the VGG19 model are as follows:

ARCHITECTURE:

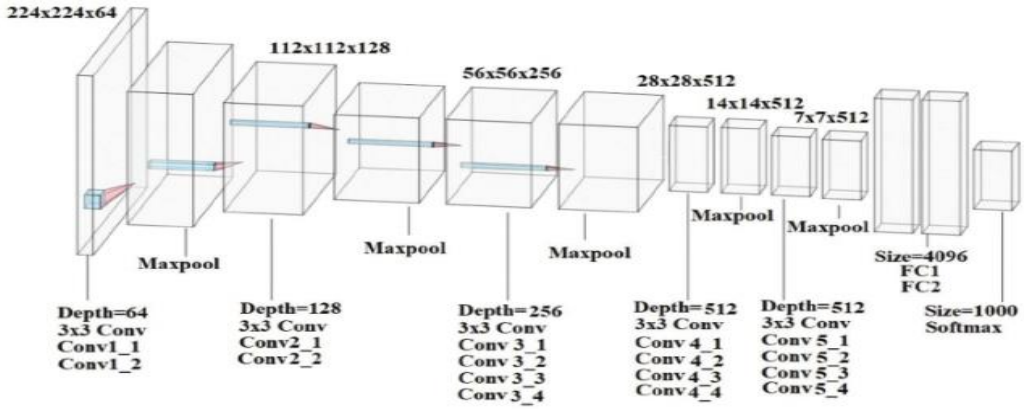

Fig 1: VGG19 Architecture

2. NAS Net Large

Neural Architecture Search Network is referred to as NAS Net. NAS Net, a Google Brain project, uses an automated search procedure to find efficient neural network architecture. The NAS Net Large model's architecture is depicted in Figure 2.[7]

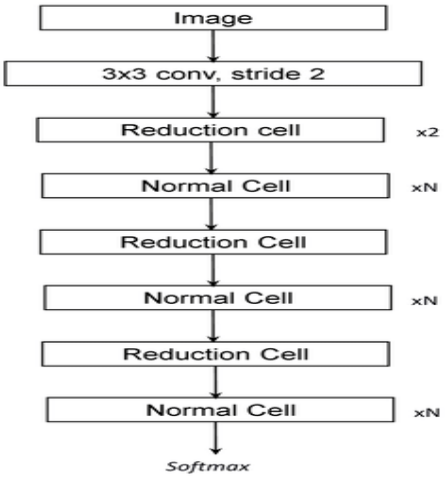

Fig 2: NAS Net Large

3. INCEPTIONRESNETV2

Based on the Inception family of architectures, Inception-ResNet-v2 is a convolutional neural architecture that adds residual connections (which take the place of the Inception design's filter concatenation stage) [8]. A convolutional neural network called Inception-ResNet-v2 was trained using over a million pictures from the ImageNet database [9]. With 164 layers, the network is capable of classifying images into 1000 different object categories, including several animals and keyboards, mouse, and pencils.

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure	35×35×288
5×Inception	As in figure	17×17×768
2×Inception	As in figure	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Table 1: Inceptionresnetv2

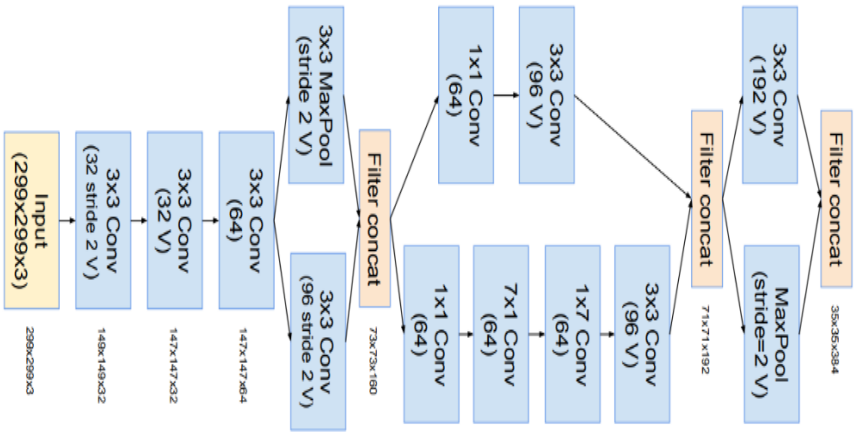

Fig 3: Inceptionresnetv2

COMPARISON ON THE BASIS OF TRANSFER LEARNING:

VGG19

After observing the architecture of Vgg19, we removed the top layer of vgg19 and then added our CNN architecture which consists of one flatten layer, one convolves layer with 256 neurons having relu as activation function and an output layer which has sigmoid as activation function. Here in the figure2 we can see the summery of the model without top layer.

```
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080
block3_conv4 (Conv2D)        (None, 37, 37, 256)       590080
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
block4_conv4 (Conv2D)        (None, 18, 18, 512)       2359808
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
block5_conv4 (Conv2D)        (None, 9, 9, 512)         2359808
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
Total params: 20024384 (76.39 MB)
Trainable params: 20024384 (76.39 MB)
Non-trainable params: 0 (0.00 Byte)
```

Fig 4: VGG19 Summary without Top Layer

```
Epoch 1/10
9/9 [==============================] - 120s 13s/step - loss: 0.8631 - accuracy: 0.6400 - val_loss: 0.4031 - val_accuracy: 0.8542
Epoch 2/10
9/9 [==============================] - 114s 13s/step - loss: 0.2433 - accuracy: 0.9164 - val_loss: 0.3909 - val_accuracy: 0.8333
Epoch 3/10
9/9 [==============================] - 114s 13s/step - loss: 0.1264 - accuracy: 0.9600 - val_loss: 0.3491 - val_accuracy: 0.8542
Epoch 4/10
9/9 [==============================] - 113s 13s/step - loss: 0.0683 - accuracy: 0.9855 - val_loss: 0.3565 - val_accuracy: 0.8333
Epoch 5/10
9/9 [==============================] - 110s 12s/step - loss: 0.0378 - accuracy: 0.9964 - val_loss: 0.3752 - val_accuracy: 0.8333
Epoch 6/10
9/9 [==============================] - 114s 13s/step - loss: 0.0254 - accuracy: 1.0000 - val_loss: 0.3824 - val_accuracy: 0.8750
Epoch 7/10
9/9 [==============================] - 111s 12s/step - loss: 0.0186 - accuracy: 1.0000 - val_loss: 0.3793 - val_accuracy: 0.8750
Epoch 8/10
9/9 [==============================] - 110s 12s/step - loss: 0.0152 - accuracy: 1.0000 - val_loss: 0.3831 - val_accuracy: 0.8542
Epoch 9/10
9/9 [==============================] - 113s 13s/step - loss: 0.0113 - accuracy: 1.0000 - val_loss: 0.3856 - val_accuracy: 0.8750
Epoch 10/10
9/9 [==============================] - 113s 13s/step - loss: 0.0087 - accuracy: 1.0000 - val_loss: 0.4021 - val_accuracy: 0.8542
```

Fig 5: Shows the detail of the training of new generated model after transfer learning for 10 epochs

Accuracy Plots Following images Fig 6. Red line plots the accuracy while training and blue line shows the accuracy while validating. Loss Plots Following images Fig 7:. Red line plots the loss while training and blue line shows the loss while validating.

```
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'],color='red',label='train')
plt.plot(history.history['val_accuracy'],color='blue',label='validation')
plt.legend()
plt.show()
```

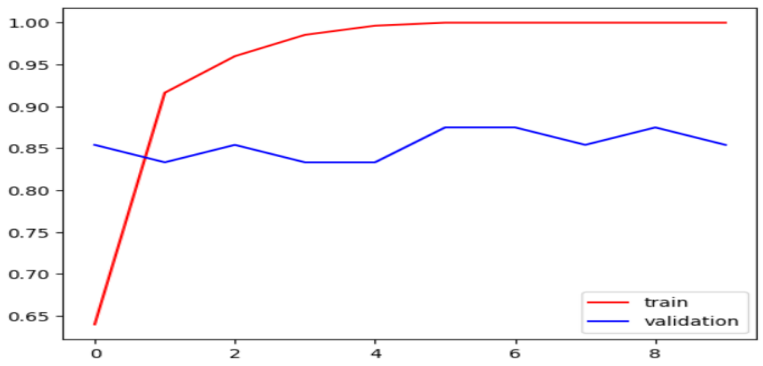

Fig 6: Shows accuracy while training and testing after every epoch.

```
plt.plot(history.history['loss'],color='red',label='train')
plt.plot(history.history['val_loss'],color='blue',label='validation')
plt.legend()
plt.show()
```

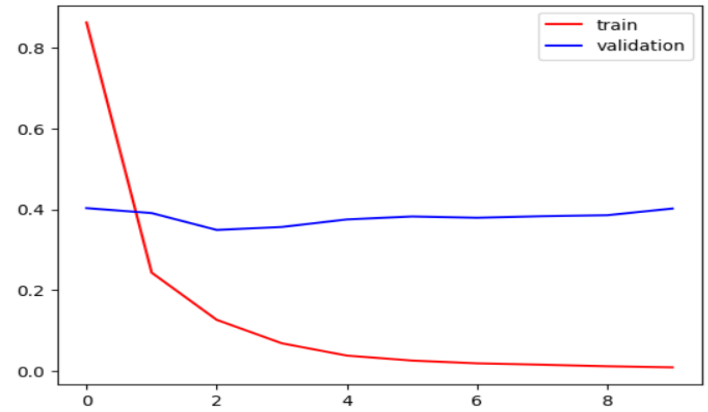

Fig 7: Shows loss while training and testing after every epoch.

After observing the architecture of NASNetLarge, we removed the top layer of NASNetLarge and then added our CNN architecture which consists of one flatten layer, one convolves layer with 256 neurons having relu as activation function and a output layer which has sigmoid as activation function. Here in the figure6 we can see the summery of the model without top layer.

```
normal_add_1_18 (Add)        (None, 5, 5, 672)         0         ['separable_conv_2_bn_normal_left1_18[0][0]',
                                                                   'separable_conv_2_bn_normal_right1_18[0][0]']

normal_add_2_18 (Add)        (None, 5, 5, 672)         0         ['separable_conv_2_bn_normal_left2_18[0][0]',
                                                                   'separable_conv_2_bn_normal_right2_18[0][0]']

normal_add_3_18 (Add)        (None, 5, 5, 672)         0         ['normal_left3_18[0][0]',
                                                                   'adjust_bn_18[0][0]']

normal_add_4_18 (Add)        (None, 5, 5, 672)         0         ['normal_left4_18[0][0]',
                                                                   'normal_right4_18[0][0]']

normal_add_5_18 (Add)        (None, 5, 5, 672)         0         ['separable_conv_2_bn_normal_left5_18[0][0]',
                                                                   'normal_bn_1_18[0][0]']

normal_concat_18 (Concatenate (None, 5, 5, 4032)       0         ['adjust_bn_18[0][0]',
                                                                   'normal_add_1_18[0][0]',
                                                                   'normal_add_2_18[0][0]',
                                                                   'normal_add_3_18[0][0]',
                                                                   'normal_add_4_18[0][0]',
                                                                   'normal_add_5_18[0][0]']

activation_259 (Activation)  (None, 5, 5, 4032)        0         ['normal_concat_18[0][0]']

==================================================================================================
Total params: 84916818 (323.93 MB)
Trainable params: 84720150 (323.18 MB)
Non-trainable params: 196668 (768.23 KB)
```

Fig 8: Summary of the model without top layer

Transfer learning with NASNetLarge architecture: Fig 9 shows the detail of the training of new generated model after transfer learning for 10 epochs:

```
Epoch 1/10
9/9 [==============================] - 149s 13s/step - loss: 3.3953 - accuracy: 0.8655 - val_loss: 2.7479 - val_accuracy: 0.9583
Epoch 2/10
9/9 [==============================] - 101s 11s/step - loss: 3.0692 - accuracy: 0.9455 - val_loss: 0.4131 - val_accuracy: 0.9375
Epoch 3/10
9/9 [==============================] - 101s 11s/step - loss: 1.2862 - accuracy: 0.9527 - val_loss: 0.4905 - val_accuracy: 0.9375
Epoch 4/10
9/9 [==============================] - 107s 12s/step - loss: 1.8639e-06 - accuracy: 1.0000 - val_loss: 0.4000 - val_accuracy: 0.9792
Epoch 5/10
9/9 [==============================] - 109s 12s/step - loss: 7.2427e-06 - accuracy: 1.0000 - val_loss: 0.7632 - val_accuracy: 0.9792
Epoch 6/10
9/9 [==============================] - 114s 12s/step - loss: 7.7974e-04 - accuracy: 1.0000 - val_loss: 0.8489 - val_accuracy: 0.9583
Epoch 7/10
9/9 [==============================] - 101s 11s/step - loss: 9.6893e-06 - accuracy: 1.0000 - val_loss: 0.8947 - val_accuracy: 0.9583
Epoch 8/10
9/9 [==============================] - 103s 11s/step - loss: 4.8126e-06 - accuracy: 1.0000 - val_loss: 0.9122 - val_accuracy: 0.9583
Epoch 9/10
9/9 [==============================] - 103s 11s/step - loss: 3.7435e-06 - accuracy: 1.0000 - val_loss: 0.9180 - val_accuracy: 0.9583
Epoch 10/10
9/9 [==============================] - 112s 13s/step - loss: 3.3785e-06 - accuracy: 1.0000 - val_loss: 0.9191 - val_accuracy: 0.9583
```

Fig 9: Training the model

Accuracy Plots Following Images Fig 10. Red line plots the accuracy while training and blue line shows the accuracy while validating. Loss Plots Following images Fig 11. Red line plots the loss while training and blue line shows the loss while validating.

```
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'],color='red',label='train')
plt.plot(history.history['val_accuracy'],color='blue',label='validation')
plt.legend()
plt.show()
```

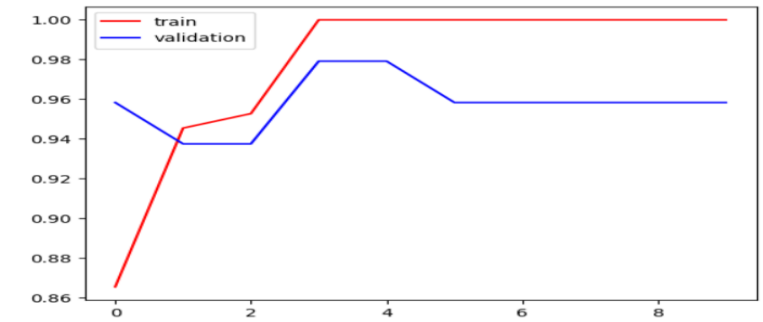

Fig 10: Shows accuracy while training and testing after every epoch.

```
plt.plot(history.history['loss'],color='red',label='train')
plt.plot(history.history['val_loss'],color='blue',label='validation')
plt.legend()
plt.show()
```

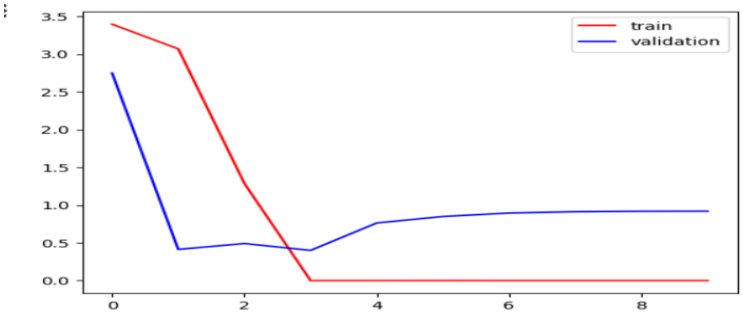

Fig 11: Shows loss while training and testing after every epoch.

We have seen the architecture of InceptionResNetV2, we removed the top layer of InceptionResNetV2 and then added our CNN architecture which consists of one flatten layer, one convolves layer with 256 neurons having relu as activation function and a output layer which has sigmoid as activation function. Here in the **Fig 12** we can see the summery of the model without top layer.

conv2d_202 (Conv2D)	(None, 3, 3, 256)	172032	['activation_461[0][0]']
batch_normalization_199 (BatchNormalization)	(None, 3, 3, 192)	576	['conv2d_199[0][0]']
batch_normalization_202 (BatchNormalization)	(None, 3, 3, 256)	768	['conv2d_202[0][0]']
activation_459 (Activation)	(None, 3, 3, 192)	0	['batch_normalization_199[0][0]']
activation_462 (Activation)	(None, 3, 3, 256)	0	['batch_normalization_202[0][0]']
block8_10_mixed (Concatenate)	(None, 3, 3, 448)	0	['activation_459[0][0]', 'activation_462[0][0]']
block8_10_conv (Conv2D)	(None, 3, 3, 2080)	933920	['block8_10_mixed[0][0]']
custom_scale_layer_39 (CustomScaleLayer)	(None, 3, 3, 2080)	0	['block8_9_ac[0][0]', 'block8_10_conv[0][0]']
conv_7b (Conv2D)	(None, 3, 3, 1536)	3194880	['custom_scale_layer_39[0][0]']
conv_7b_bn (BatchNormalization)	(None, 3, 3, 1536)	4608	['conv_7b[0][0]']
conv_7b_ac (Activation)	(None, 3, 3, 1536)	0	['conv_7b_bn[0][0]']

```
Total params: 54336736 (207.28 MB)
Trainable params: 54276192 (207.05 MB)
Non-trainable params: 60544 (236.50 KB)
```

Fig 12: Summary of the model without top layer

```
9/9 [==============================] - 62s 5s/step - loss: 2.0347 - accuracy: 0.8655 - val_loss: 2.2550 - val_accuracy: 0.9167
Epoch 2/10
9/9 [==============================] - 36s 4s/step - loss: 1.5337 - accuracy: 0.9564 - val_loss: 0.9073 - val_accuracy: 0.9375
Epoch 3/10
9/9 [==============================] - 36s 4s/step - loss: 0.8890 - accuracy: 0.9491 - val_loss: 0.3689 - val_accuracy: 0.9792
Epoch 4/10
9/9 [==============================] - 35s 4s/step - loss: 0.4502 - accuracy: 0.9745 - val_loss: 0.6589 - val_accuracy: 0.9167
Epoch 5/10
9/9 [==============================] - 41s 4s/step - loss: 0.1339 - accuracy: 0.9927 - val_loss: 1.7777 - val_accuracy: 0.9167
Epoch 6/10
9/9 [==============================] - 35s 4s/step - loss: 0.0431 - accuracy: 0.9964 - val_loss: 0.4580 - val_accuracy: 0.9583
Epoch 7/10
9/9 [==============================] - 35s 4s/step - loss: 0.0841 - accuracy: 0.9927 - val_loss: 0.4461 - val_accuracy: 0.9583
Epoch 8/10
9/9 [==============================] - 37s 4s/step - loss: 0.0178 - accuracy: 0.9964 - val_loss: 1.3136 - val_accuracy: 0.9375
Epoch 9/10
9/9 [==============================] - 35s 4s/step - loss: 2.3448e-07 - accuracy: 1.0000 - val_loss: 0.5811 - val_accuracy: 0.9375
Epoch 10/10
9/9 [==============================] - 40s 4s/step - loss: 7.3028e-13 - accuracy: 1.0000 - val_loss: 0.3970 - val_accuracy: 0.9375
```

Fig 13: Training the model'

```
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'],color='red',label='train')
plt.plot(history.history['val_accuracy'],color='blue',label='validation')
plt.legend()
plt.show()
```

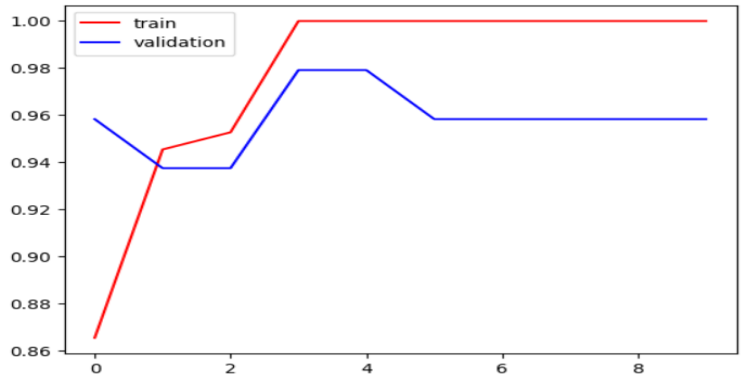

Fig 14: Shows accuracy while training and testing after every epoch.

```
plt.plot(history.history['loss'],color='red',label='train')
plt.plot(history.history['val_loss'],color='blue',label='validation')
plt.legend()
plt.show()
```

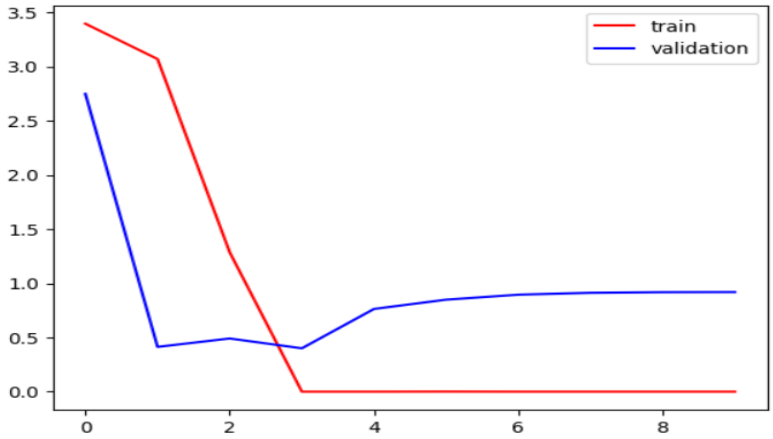

Fig 15: Shows loss while training and testing after every epoch.

3. CONCLUSION

Our paper demonstrated the result of the comparison among three pretrained architecture VGG19, NASNetLarge and InceptionResNetV2 of the performance after transfer learning.

Table 2 Shows the detail of the comparison.

Architecture	Loss	Accuracy	Val_Loss	Val_Accuracy
VGG19	0.0087	1.00	0.4021	0.8542
NASNetLarge	3.37	1.00	0.9191	0.9583
InceptionResNetV2	7.3028e-13	1.00	0.3970	0.9375

As Table 2 shows VGG19 gives Loss of 0.0087 on training data in transfer learning while InceptionResNetV2 gives the loss of 7.3028e-13 which is smallest among three on training data. Loss on validating data is almost same in both the architectures. But NAS Net Large gives the highest loss on validating data. As we can see Val_Accuracy is smallest in case of VGG19.

4. REFERENCES

1. A Convolutional Neural Network Classifier VGG-19 Architecture for Lesion Detection and Grading in Diabetic Retinopathy Based on Deep Learning, V. Sudha and T. R. Ganeshbabu, 10 June 2020; Accepted: 16 July 2020, Tech Science Press, Materials & Continua 66(1):827-842, DOI:10.32604/cmc.2020.012008
2. Pay Attention to Features, Transfer Learn Faster CNNs Kafeng Wang 1, Xitong Gao2, Yiren Zhao3, Xingjian Li4, Dejing Dou5, Cheng-Zhong Xu6, Published as a conference paper at ICLR 2020.
3. Deep Learning and Transfer Learning Approaches for Image Classification. Sajja Tulasi Krishna, Hemantha Kumar Kalluri, International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-7, Issue-5S4, February 2019
4. K. Weiss, T.M. Khoshgoftaar, and D. Wang, "A survey of transferlearning," J. Big Data, vol. 3, no. 1, Dec. 2016.
5. Transfer learning and its application research Yancong Zhou, Xuemei Zhang, Yan Wang, Bo Zhang EICE 2021 Journal of Physics: Conference Series, 1920 (2021) 012058, IOP Publishing, doi:10.1088/1742-6596/1920/1/012058
6. Transfer learning for image classification using VGG19: Caltech-101 image data set, Original Research, Published: 17 September 2021, Volume 14, pages 3609–3620, (2023)
7. Detection of coronavirus disease from X-ray images using deep learning and transfer learning algorithms Saleh Albahli and Waleed Albattah, August 2020Journal of X-Ray Science and Technology 28(5), DOI:10.3233/XST-200720, LicenseCC BY-NC 4.0
8. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi, https://doi.org/10.48550/arXiv.1602.07261
9. Deep convolution neural networks to differentiate between COVID-19 and other pulmonary abnormalities on chest radiographs: Evaluation using internal and external datasets, Yongwon Cho, Sung Ho Hwang, Yu-Whan Oh, Beom Jin Park, VL - 31, DOI - 10.1002/ima.22595, JO - International Journal of Imaging Systems and Technology
10. Layers of vgg19 from https://iq.opengenus.org/vgg19-architecture