DATA ANALYTICS – PROJECT REPORT

NAME: VENKATA SIVA KIRAN PILLALAMARRI

IMAGE CLASSIFICATION

INTRODUCTION:

The ultimate objective of image classification is to create models that properly classify images into multiple groups. Object recognition, identification of faces, analysis of medical images, real-world applications are just a few of the many uses for image categorization, a core problem in computer vision.

A machine learning method is used to examine an image's visual characteristics and determine the best label or category for it. This is the basic notion underlying image classification. The system is trained on a sizable dataset of labeled photos in order to achieve this, and it is then applied to new, unlabeled images to provide predictions. By tweaking the algorithm's parameters, utilizing more sophisticated architectures, or expanding the training data, the algorithm's accuracy can be increased.

There are 16 distinct kinds of classes, and the aim is to use the data that has been provided to build a model that can correctly categorize these images. The performance of the model must be assessed once it has been trained using a validation set. To measure how successfully the model can categorize images, different performance metrics including accuracy, precision, recall, and F1-score are calculated. Ultimately, the model that was trained may be applied when predicting image class labels from a test set. To evaluate the model's accuracy and generalizability, the predictions may be compared to the actual class labels. In general, picture categorization is a potent tool that can assist us in making sense of the enormous amounts of visual information we come across daily.

DATA:

We are given 16 different classes of images as test and training data in which we need to use only the training data consisting of 1838 images of different formats (Jpeg, Jpg, Png). Considering the training data for the building and training of model the total 1838 images are divided into train and valid data like 70% as train data and 30% data as valid data. The test data will be used only for testing it should not be used in training of the model. After building the model, we need to evaluate the model on the valid data to test for validation.

Then the accuracy score and the precision, recall, and F1-score are calculated. Here we can use different classification algorithms such as CNN, SVM, RF.

Classes: Adult, Airplane, Alpaca, Bird. Bus, Car, Cat, Child, Elephant, Flower, Giraffe, Horse, Monkey, Panda, Reptile, Vessel.

PERFORMED ANALYSIS:

Data Preparation: The image data must first be prepared by being split into a training set and a validation set. Additionally, the data should be standardized and scaled to a standard size.

Data Augmentation: Techniques for enhancing the training set's size and avoiding overfitting can be applied. This entails making modifications to the photos, including flipping, rotating, zooming, shifting.

A framework like TensorFlow or Keras may be used to create the CNN model. Convolutional, pooling, and fully linked layers are among the layers that should be present in the model.

Model Compilation: After the model has been constructed, it must be compiled by defining the loss function, optimizer, metrics.

Training the Model: Using the fit () method, the model may then be trained using the training set. The model's weights are updated during training based on loss function and optimizer.

Model Evaluation: After training, the model must be assessed using the evaluate () function on the validation set. This is a prediction of the model's performance on fresh, untested data.

Making Predictions: Lastly, the predict () function of the model may be used to create predictions based on fresh, previously unobserved data. The predictions may be checked with the true labels to determine the model's accuracy.

CNN (Convolutional Neural Networks):

Basically, here at first, I built CNN model in which I have taken the train directory and as we are given with only the classes I assigned the labels for each class like 0 for adult and 1 for airplane and 2 for bird and so on. Here I use the data augmentation in order to achieve best results.

After the data preparation it convert the labels and images to NumPy and move forward to build the model. Here I used 4 Conv2D layers, 4 MaxPooling2D layers, 1 Flatten layer, 2 Dense layers, and 1 Dropout layer, Dense layer with 512

units and ReLU activation, Dropout layer with rate of 0.5, Dense layer with 16 units and SoftMax activation.

After that 'Adam' is an optimizer. The network weights are updated using this well-liked stochastic gradient descent (SGD) optimization technique based on the gradients of the loss function. Categorical cross entropy loss, when one-hot encoding is utilized for the output labels, this kind of loss function is employed for multi-class classification issues.

Metrics: 'accuracy' The measurement to be used for assessment during training and testing are specified here. The statistic in this situation is accuracy, which is the percentage of properly categorized images.

CNN-PRETRAINED:

This method uses transfer learning using a pre-trained Mobile Net model to create a model for classifying images in 16 different classes. The picture data generators are set up for the validation and training datasets in the first section of the algorithm. The training set and validation set are divided into two groups with a 90:10 ratio when the data is loaded via the "Training Image sp 23" directory.

Several modifications, including shear, zoom, and horizontal flip, are also used to enhance the training data. The pre-trained Mobile Net algorithm is then loaded, the top layers are deleted, and additional layers are added to produce a new model tailored to our particular purpose.

Because the weights of the layers in the base model have been frozen, they will not be changed during training, and just the weights of the newly added layers will be trained.

An accuracy metric, categorical cross-entropy loss function, and Adam optimizer are then used to construct the model. In the next 10 iterations, the model is trained using the training data and verified using the validation data. Each epoch records the accuracy and loss for training and validation. To acquire the validation accuracy, that is reported to the console, the model is finally assessed using the validation data.

SVM (Support Vector Machines):

The dataset should be loaded, then split into training and valid sets. Define the SVM model by doing the following: The hyperparameters should be defined after importing the SVM model from the Scikit-Learn package.

The regularization parameter (C), gamma value, and kernel type are examples of hyperparameters. Use the feature extraction technique LBP as described in below. Model training: Using the fit () technique, train the SVM model using the training set. Evaluate the model: Use the predict () function to assess the SVM model on the valid set, and the accuracy score () method to obtain the accuracy score.

The SVM model's performance may be enhanced by tuning the hyperparameters, if necessary. Methods like cross-validation and grid search can be used for this. Once the model is built perform the model on the new test data in order to get the test accuracy and well the model is performing.

LBP-LOCAL BINARY PATTERN:

For imagine classification tasks, a texture descriptor called Local Binary Pattern (LBP) is utilized. Each pixel's intensity value is compared to the intensity values of the pixels around it, and the result is encoded as a binary integer. The image's texture pattern is then described using this binary code as a feature.

The image's pixels are individually surrounded by a 3x3 window that is centered on them. Each of the center pixel's eight neighbors' intensity values are contrasted with it. In the binary code, the neighbor's location is assigned a "1" if the intensity value of the neighbor is higher than or equal to the intensity value of the center pixel, and a "0" otherwise. Each comparison's results in clockwise order are concatenated to create the binary code.

The binary code that results is translated to decimal, and the decimal value is utilized to determine the LBP feature for that pixel. The radius and the total number of neighbors are the LBP algorithm's two parameters. The radius establishes the size of the circular area surrounding the center pixel, & the number of neighbors establishes the frequency of neighbor comparisons with the central pixel. An 8-bit binary code is produced when these parameters are commonly set to 1 and 8, respectively.

SVM-PRETRAINED-VGG16:

After I used VGG16 which was pretrained model such that the top layers are freeze in code we write the include top= false which tells that top layers are not used, and weights are ImageNet is used the last fully connected layer is in responsible for generating classification probabilities. When include top is set to False, the pre-trained model ignores this layer, which is beneficial for feature extraction.

The hyperparameters should be defined after importing the SVM model from the Scikit-Learn package. The regularization parameter (C), gamma value, and kernel type are examples of hyperparameters. Model training: Using the fit () technique, train the SVM model using the training set.

Evaluate the model: Use the predict () function to assess the SVM model on the valid set, and the accuracy score () method to obtain the accuracy score.

The SVM model's performance may be enhanced by tuning the hyperparameters, if necessary. Methods like cross-validation and grid search can be used for this. Once the model is built perform the model on the new test data in order to get the test accuracy and well the model is performing. BY using this VGG 16 pretrained method I got change of accuracy from 45%(NORMAL) to 84% accuracy.

RF (RANDOM FOREST CLASSIFIER):

The dataset should be loaded, then split into training and valid sets. After I discussed in above LBP as my feature extraction technique with the train data ad tune the remaining parameters as I discussed in below.

Define the RF model by doing the following: The hyperparameters should be defined after importing the RF model from the Scikit-Learn package.

The number of trees in the forest (n_estimator's), the maximum depth of each tree (max_depth), and the lowest number of samples needed to divide an internal node (min_samples_split) are a few crucial hyperparameters to consider.

Model training: Using the fit () technique, train the RF model using the training set. Evaluate the model: Use the predict () function to assess the RF model on the valid set, and the accuracy score () method to obtain the accuracy score.

The RF model's performance may be enhanced by tuning the hyperparameters, if necessary. Methods like cross-validation and grid search can be used for this. Once the model is built perform the model on the new test data in order to get the test accuracy and well the model is performing. Once the model is built perform the model on the new test data in order to get the test accuracy and well the model is performing. BY using this LBP method, I got change of accuracy from 27% to 40%.

COMPARISION OF SVM, RF, CNN:

CNNs are an advanced deep learning model that is often used in computer vision applications. SVMs are a versatile machine learning algorithm that is used for

classification and regression. Finally, RF is an ensemble learning approach that combines decision trees to increase accuracy and resilience. Basically, in order to tell which model is best model for image classification will totally depend on 4 parameters. They are.

1. Model architecture

2. Training and prediction

3.Interpretability

4.Performance

Model architecture:

SVM is a linear model with the goal to determine the optimum hyperplane that divides the data into different classes. CNN is a deep learning technique that employs convolutional layers to acquire features from images. RF is an ensemble learning approach that predicts by combining many decision trees.

Training and prediction:

More quickly than CNN and RF, SVM can be trained and predicted. For tasks requiring image and audio recognition, CNN may be more successful than SVM and RF but may require a longer training period. RF may require less time to train than CNN, but it may not be as accurate with large datasets.

Interpretability:

SVM and RF models are often more interpretable than CNN models since they employ displayed and analyzed decision boundaries and tree topologies. CNN models include complicated layers and feature maps that might be difficult to decipher.

Performance:

In general, CNNs excel at tasks involving image and audio recognition, whereas SVMs and RFs can excel across a variety of datasets and tasks.

RESULTS AND DISCUSSION:

ANALYSIS OF OBSERVED RESULTS:

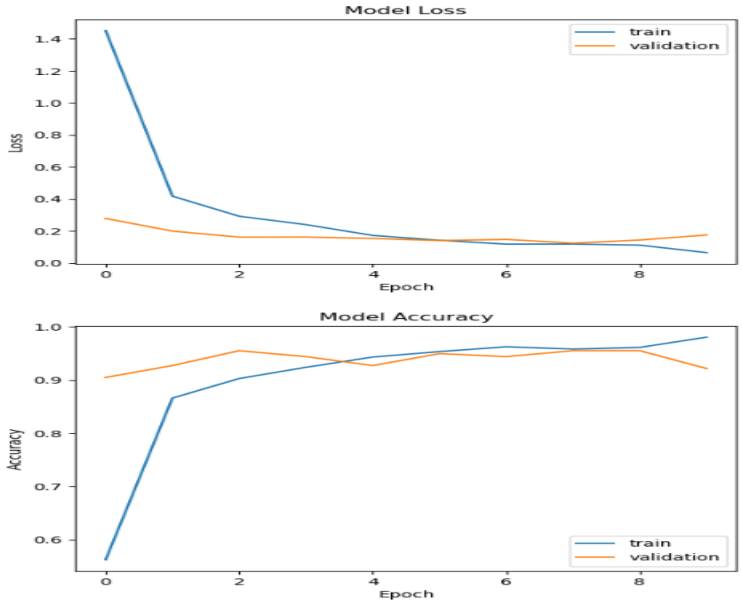
The CNN pretrained utilizing MobileNet looks to be the highest performing model for the image classification test, with an accuracy of 94%. This is an enormous gain above the other models i evaluated, which included CNN no pretraining, SVM plus LBP, and RF using LBP, all of which had accuracies 48,43,38 %. While this is a

commendable performance, the CNN with MobileNet still outperformed the SVM model via VGG features with an accuracy of 87%. With an accuracy of 82%, the RF model utilizing VGG features additionally performed reasonably well.

Overall, my findings imply that, when compared to more conventional machine learning techniques like SVM and RF, deep learning models like CNNs that have been pretrained on big datasets may be quite successful at performing image classification tasks. In order to perform better on smaller, more focused tasks, trained models might take advantage of the rich representations learnt from huge datasets. It's also important to note that the efficiency of both the SVM as well as the RF models was significantly impacted by the feature extraction approach used (LBP vs VGG), with VGG features often producing greater accuracy results. This emphasizes how crucial it is to carefully choose the best feature extraction methods for a particular assignment. In conclusion, the CNN pretrained utilizing MobileNet model had the highest performance for the image classification test, obtaining a 94% accuracy. This demonstrates the utility of trained models for image categorization tasks.

CNN-PRETRAINED: TRAINING AND VALIDATION

```
Epoch 5/10
52/52 [==============================] - 38s 736ms/step - loss: 0.1723 - accuracy: 0.9433 - val_loss: 0.1534 - val_accuracy: 0.
9274
Epoch 6/10
52/52 [==============================] - 38s 738ms/step - loss: 0.1420 - accuracy: 0.9536 - val_loss: 0.1402 - val_accuracy: 0.
9497
Epoch 7/10
52/52 [==============================] - 38s 738ms/step - loss: 0.1179 - accuracy: 0.9626 - val_loss: 0.1473 - val_accuracy: 0.
9441
Epoch 8/10
52/52 [==============================] - 39s 740ms/step - loss: 0.1181 - accuracy: 0.9584 - val_loss: 0.1240 - val_accuracy: 0.
9553
Epoch 9/10
52/52 [==============================] - 38s 732ms/step - loss: 0.1116 - accuracy: 0.9614 - val_loss: 0.1436 - val_accuracy: 0.
9553
Epoch 10/10
52/52 [==============================] - 38s 738ms/step - loss: 0.0649 - accuracy: 0.9807 - val_loss: 0.1751 - val_accuracy: 0.
9218
```



```
6/6 [==============================] - 4s 600ms/step - loss: 0.1389 - accuracy: 0.9553
Validation Accuracy: 0.9553072452545166
```

CNN-PRETRAINED: TESTING

```
# Print the test accuracy
print('Test Accuracy:', accuracy)
```

```
15/15 [==============================] - 7s 443ms/step - loss: 0.1691 - accuracy: 0.9458
Test Accuracy: 0.9458333253860474
```

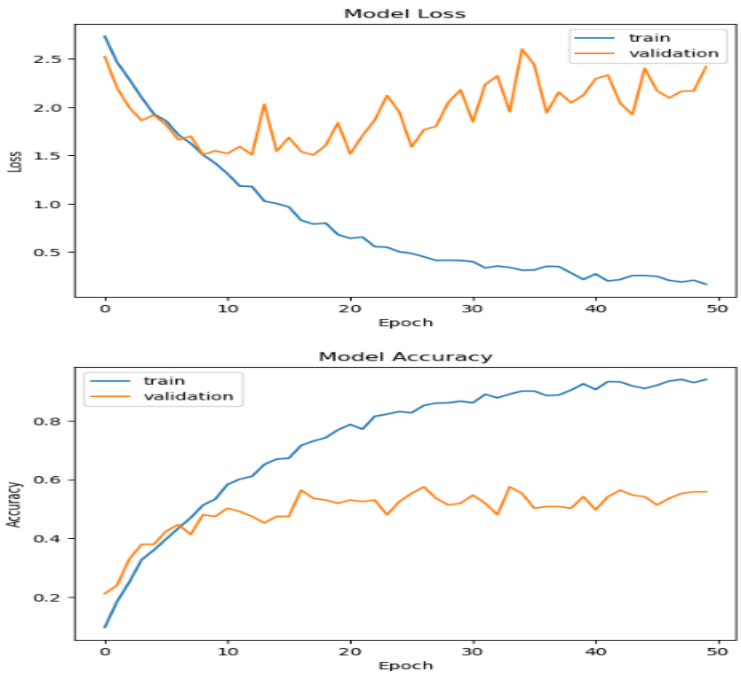
```
print(classification_report(test_labels, predicted_labels))
```

	precision	recall	f1-score	support
0	0.95	0.63	0.76	30
1	0.97	1.00	0.98	30
2	0.86	1.00	0.92	30
3	0.97	1.00	0.98	30
4	1.00	1.00	1.00	30
5	0.97	1.00	0.98	30
6	1.00	0.90	0.95	30
7	0.67	0.97	0.79	30
8	0.97	1.00	0.98	30
9	1.00	0.97	0.98	30
10	1.00	0.87	0.93	30
11	1.00	0.87	0.93	30
12	1.00	0.97	0.98	30
13	1.00	0.97	0.98	30
14	0.97	1.00	0.98	30
15	0.97	1.00	0.98	30
accuracy			0.95	480
macro avg	0.96	0.95	0.95	480
weighted avg	0.96	0.95	0.95	480

```
cm = confusion_matrix(test_labels, predicted_labels)
print(cm)
```

```
[[19  0  0  0  0  0  0 11  0  0  0  0  0  0  0  0]
 [ 0 30  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0 30  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 30  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0 30  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 30  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  1  0  0 27  2  0  0  0  0  0  0  0  0]
 [ 1  0  0  0  0  0  0 29  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 30  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 29  0  0  0  0  1  0]
 [ 0  1  3  0  0  0  0  0  0  0 26  0  0  0  0  0]
 [ 0  0  1  0  0  1  0  0  1  0  0 26  0  0  0  1]
 [ 0  0  1  0  0  0  0  0  0  0  0  0 29  0  0  0]
 [ 0  0  0  0  0  0  0  1  0  0  0  0  0 29  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0 30  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 30]]
```

CNN-NORMAL: TRAINING AND VALIDATION



```
52/52 [==============================] - 150s 3s/step - loss: 0.2591 - accuracy: 0.9102 - val_loss: 2.4040 - val_accuracy:
0.5419
Epoch 46/50
52/52 [==============================] - 65s 1s/step - loss: 0.2496 - accuracy: 0.9210 - val_loss: 2.1708 - val_accuracy: 0.
5140
Epoch 47/50
52/52 [==============================] - 65s 1s/step - loss: 0.2079 - accuracy: 0.9355 - val_loss: 2.0967 - val_accuracy: 0.
5363
Epoch 48/50
52/52 [==============================] - 65s 1s/step - loss: 0.1911 - accuracy: 0.9409 - val_loss: 2.1660 - val_accuracy: 0.
5531
Epoch 49/50
52/52 [==============================] - 66s 1s/step - loss: 0.2092 - accuracy: 0.9301 - val_loss: 2.1699 - val_accuracy: 0.
5587
Epoch 50/50
52/52 [==============================] - 66s 1s/step - loss: 0.1680 - accuracy: 0.9409 - val_loss: 2.4193 - val_accuracy: 0.
5587
```

CNN-NORMAL: TESTING

```
# Print the test accuracy
print('Test Accuracy:', accuracy*100)
```

```
15/15 [==============================] - 7s 456ms/step - loss: 3.4775 - accuracy: 0.4729
Test Accuracy: 47.29166626930237
```

```
]: from sklearn.metrics import classification_report

print(classification_report(test_labels, predicted_labels))
```

	precision	recall	f1-score	support
0	0.25	0.43	0.32	30
1	0.48	0.53	0.51	30
2	0.42	0.33	0.37	30
3	0.27	0.13	0.18	30
4	0.49	0.80	0.61	30
5	0.44	0.50	0.47	30
6	0.36	0.30	0.33	30
7	0.48	0.43	0.46	30
8	0.55	0.37	0.44	30
9	0.54	0.50	0.52	30
10	0.40	0.57	0.47	30
11	0.74	0.47	0.57	30
12	0.40	0.27	0.32	30
13	0.64	0.83	0.72	30
14	0.50	0.27	0.35	30
15	0.66	0.83	0.74	30
accuracy			0.47	480
macro avg	0.48	0.47	0.46	480
weighted avg	0.48	0.47	0.46	480

```
cm = confusion_matrix(test_labels, predicted_labels)
print(cm)
```

```
[[13  0  1  1  1  2  4  3  0  0  0  0  1  1  1  2]
 [ 1 16  0  1  3  3  0  0  0  0  0  0  0  0  1  5]
 [ 3  0 10  1  0  0  1  0  2  1  5  1  1  4  1  0]
 [ 4  6  0  4  1  2  0  1  2  2  1  2  1  1  1  2]
 [ 1  0  0  0 24  4  0  0  0  0  0  0  0  0  0  1]
 [ 0  3  0  0  8 15  0  1  0  0  0  0  1  0  0  2]
 [ 3  0  1  3  0  0  9  4  0  3  1  0  2  1  2  1]
 [ 9  0  0  0  3  0  3 13  0  1  0  0  1  0  0  0]
 [ 0  1  4  1  0  0  0  0 11  1 11  0  0  1  0  0]
 [ 6  0  1  0  3  0  0  0  0 15  0  1  2  1  1  0]
 [ 1  1  3  0  1  1  2  2  0  1 17  0  1  0  0  0]
 [ 2  0  4  1  0  1  0  1  3  1  2 14  0  1  0  0]
 [ 3  2  0  2  0  2  3  1  2  3  1  0  8  2  1  0]
 [ 0  0  0  1  0  1  0  0  0  0  0  1  2 25  0  0]
 [ 4  1  0  0  4  3  3  1  0  0  4  0  0  2  8  0]
 [ 1  3  0  0  1  0  0  0  0  0  0  0  0  0  0 25]]
```

SVM-PRETRAINED: TRAINING AND VALIDATION

```
print(classification_report(valid_labels, valid_pred))
```

	precision	recall	f1-score	support
0	0.57	0.89	0.70	18
1	0.88	0.92	0.90	24
2	0.90	0.78	0.84	23
3	1.00	0.74	0.85	19
4	0.95	0.83	0.88	23
5	0.85	0.85	0.85	27
6	0.94	0.89	0.91	36
7	0.67	0.31	0.42	13
8	1.00	0.90	0.95	21
9	0.86	1.00	0.92	18
10	0.82	0.82	0.82	28
11	1.00	0.82	0.90	17
12	0.73	0.88	0.80	25
13	0.93	0.89	0.91	28
14	0.77	0.74	0.76	23
15	0.73	0.96	0.83	25
accuracy			0.84	368
macro avg	0.85	0.83	0.83	368
weighted avg	0.86	0.84	0.84	368

```
: print(f"Validation accuracy: {valid_acc}")
```

```
Validation accuracy: 0.842391304347826
```

SVM-PRETRAINED: TESTING

	precision	recall	f1-score	support
0	0.71	0.83	0.77	30
1	0.97	0.93	0.95	30
2	0.83	0.80	0.81	30
3	0.85	0.73	0.79	30
4	1.00	1.00	1.00	30
5	1.00	1.00	1.00	30
6	0.81	0.73	0.77	30
7	0.79	0.77	0.78	30
8	0.89	0.83	0.86	30
9	1.00	0.93	0.97	30
10	0.76	0.87	0.81	30
11	0.92	0.77	0.84	30
12	0.71	0.80	0.75	30
13	0.82	0.93	0.87	30
14	0.86	0.83	0.85	30
15	0.91	1.00	0.95	30
accuracy			0.86	480
macro avg	0.86	0.86	0.86	480
weighted avg	0.86	0.86	0.86	480

```
cm = confusion_matrix(test_labels,test_pred)
print(cm)
```

```
[[25  0  0  0  0  0  0  4  0  0  0  0  1  0  0  0]
 [ 1 28  0  0  0  0  0  0  0  0  0  0  0  0  0  1]
 [ 0  0 24  0  0  0  0  0  1  0  2  2  1  0  0  0]
 [ 0  1  0 22  0  0  0  0  0  0  1  0  3  0  2  1]
 [ 0  0  0  0 30  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 30  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  1  0  0 22  1  0  0  0  0  1  4  1  0]
 [ 5  0  0  0  0  0  1 23  0  0  0  0  0  1  0  0]
 [ 0  0  2  0  0  0  0  0 25  0  2  0  1  0  0  0]
 [ 1  0  0  0  0  0  1  0  0 28  0  0  0  0  0  0]
 [ 0  0  1  0  0  0  2  0  0  0 26  0  0  0  0  1]
 [ 2  0  0  1  0  0  1  0  2  0  0 23  0  1  0  0]
 [ 1  0  2  0  0  0  0  0  0  0  2  0 24  0  1  0]
 [ 0  0  0  0  0  0  0  1  0  0  1  0  0 28  0  0]
 [ 0  0  0  2  0  0  0  0  0  0  0  0  3  0 25  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 30]]
```

```
print(f"test accuracy: {test_acc}")
```

```
test accuracy: 0.8604166666666667
```

SVM-NORMAL: TRAINING AND VALIDATION

```
val_accuracy = accuracy_score(y_valid, y_pred)
print('Accuracy:', val_accuracy*100)
```

```
Accuracy: 48.641304347826086
```

SVM-NORMAL: TESTING

	precision	recall	f1-score	support
Adult	0.22	0.30	0.25	30
Airplane	0.53	0.67	0.59	30
Alpaca	0.43	0.67	0.52	30
Bird	0.28	0.17	0.21	30
Bus	0.89	0.80	0.84	30
Car	0.62	0.53	0.57	30
Cat	0.27	0.40	0.32	30
Child	0.29	0.17	0.21	30
Elephant	0.28	0.43	0.34	30
Flower	0.47	0.47	0.47	30
Giraffe	0.60	0.40	0.48	30
Horse	0.45	0.17	0.24	30
Monkey	0.36	0.33	0.34	30
Panda	0.40	0.47	0.43	30
Reptile	0.48	0.37	0.42	30
Vessel	0.63	0.57	0.60	30
accuracy			0.43	480
macro avg	0.45	0.43	0.43	480
weighted avg	0.45	0.43	0.43	480

```
cm = confusion_matrix(test_labels, y_pred_test)
print(cm)
```

```
[[ 9  0  0  0  0  0  3  5  1  6  0  1  3  2  0  0]
 [ 1 20  0  5  0  0  0  0  0  1  0  0  0  0  1  2]
 [ 0  0 20  0  1  0  1  0  2  0  2  0  1  2  0  1]
 [ 2  8  1  5  0  0  6  0  1  2  1  0  1  0  1  2]
 [ 1  1  0  0 24  4  0  0  0  0  0  0  0  0  0  0]
 [ 2  4  1  1  0 16  1  0  0  0  0  0  0  0  0  5]
 [ 5  0  3  0  0  0 12  1  0  0  0  0  4  4  1  0]
 [14  0  0  0  0  4  4  5  1  0  0  0  2  0  0  0]
 [ 1  0  9  0  0  1  1  0 13  0  1  1  0  2  1  0]
 [ 2  1  0  1  0  0  2  1  2 14  0  1  5  0  1  0]
 [ 0  0  3  1  1  1  0  0  7  0 12  2  1  2  0  0]
 [ 1  0  2  0  0  0  3  1 16  1  0  5  0  0  1  0]
 [ 1  0  4  1  0  0  4  0  0  1  0  0 10  5  4  0]
 [ 2  0  1  1  0  0  5  3  1  1  0  0  0 14  2  0]
 [ 0  1  3  0  0  0  3  1  1  1  4  0  1  4 11  0]
 [ 0  3  0  3  1  0  0  0  2  3  0  1  0  0  0 17]]
```

```
print('Accuracy:', test_accuracy*100)
```

```
Accuracy: 43.125
```

RF-NORMAL: TRAINING AND VALIDATION

```
print('Validation accuracy:', val_accuracy)
```

```
Validation accuracy: 0.3858695652173913
```

RF-NORMAL: TESTING

```
print(f'Classification Report:\n{report}')
```

```
Accuracy: 0.3770833333333333
```

```
Classification Report:
              precision    recall  f1-score   support

       Adult       0.12      0.03      0.05        30
    Airplane       0.44      0.70      0.54        30
      Alpaca       0.28      0.37      0.31        30
        Bird       0.64      0.23      0.34        30
         Bus       0.50      0.90      0.64        30
         Car       0.18      0.07      0.10        30
         Cat       0.20      0.30      0.24        30
       Child       0.50      0.07      0.12        30
    Elephant       0.44      0.40      0.42        30
      Flower       0.37      0.47      0.41        30
     Giraffe       0.53      0.30      0.38        30
       Horse       0.32      0.40      0.35        30
      Monkey       0.23      0.30      0.26        30
       Panda       0.35      0.53      0.42        30
     Reptile       0.50      0.30      0.37        30
      Vessel       0.54      0.67      0.60        30

    accuracy                           0.38       480
   macro avg       0.38      0.38      0.35       480
weighted avg       0.38      0.38      0.35       480
```

```
cm = confusion_matrix(true_labels, predictions)
print(cm)
```

```
[[ 1  2  1  0  1  0  8  0  2  4  0  3  5  3  0  0]
 [ 0 21  1  1  1  0  0  0  0  3  0  0  0  0  0  3]
 [ 0  0 11  0  2  0  0  0  4  0  3  2  0  2  5  1]
 [ 0  6  1  7  0  0  7  0  0  4  0  3  1  0  0  1]
 [ 1  0  1  0 27  0  0  0  0  0  0  0  0  0  0  1]
 [ 0  8  2  0  7  2  0  0  0  0  0  4  0  0  0  7]
 [ 2  0  2  0  1  1  9  0  0  2  0  0  6  6  1  0]
 [ 3  2  2  0  1  0  7  2  0  1  1  5  3  3  0  0]
 [ 0  0  4  0  2  4  1  0 12  0  1  0  1  4  0  1]
 [ 0  2  0  0  0  0  4  0  0 14  0  2  6  1  1  0]
 [ 0  0  1  2  7  0  0  0  5  0  9  1  2  1  0  2]
 [ 0  0  7  0  1  3  0  0  1  0  1 12  2  3  0  0]
 [ 1  1  2  0  0  0  3  1  1  3  0  2  9  5  1  1]
 [ 0  0  1  0  1  0  1  1  1  0  1  4  3 16  1  0]
 [ 0  0  4  0  0  1  4  0  1  7  1  0  1  2  9  0]
 [ 0  6  0  1  3  0  0  0  0  0  0  0  0  0  0 20]]
```

RF-PRETRAINED:

```
print(f"test accuracy: {test_acc}")
```

```
test accuracy: 0.8166666666666667
```

```
print(classification_report(test_labels,test_pred))
```

	precision	recall	f1-score	support
0	0.71	0.83	0.77	30
1	0.90	0.93	0.92	30
2	0.76	0.53	0.63	30
3	0.86	0.60	0.71	30
4	0.86	1.00	0.92	30
5	0.96	0.90	0.93	30
6	0.72	0.77	0.74	30
7	0.80	0.53	0.64	30
8	0.79	0.90	0.84	30
9	0.81	1.00	0.90	30
10	0.85	0.77	0.81	30
11	0.77	0.77	0.77	30
12	0.69	0.73	0.71	30
13	0.76	0.97	0.85	30
14	0.90	0.90	0.90	30
15	0.97	0.93	0.95	30
accuracy			0.82	480
macro avg	0.82	0.82	0.81	480
weighted avg	0.82	0.82	0.81	480

```
cm = confusion_matrix(test_labels, test_pred)
print(cm)
```

```
[[25  0  0  1  0  0  0  1  0  1  0  0  2  0  0  0]
 [ 1 28  0  0  0  0  0  0  0  1  0  0  0  0  0  0]
 [ 0  0 16  0  0  0  1  0  4  0  1  5  1  1  0  1]
 [ 0  3  0 18  0  0  1  0  0  2  1  0  4  0  1  0]
 [ 0  0  0  0 30  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  3 27  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  1  0  0 23  0  1  1  0  0  0  4  0  0]
 [ 9  0  0  0  0  0  1 16  0  1  0  0  1  2  0  0]
 [ 0  0  1  0  0  0  0  0 27  0  2  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 30  0  0  0  0  0  0]
 [ 0  0  1  0  0  0  2  0  0  1 23  1  1  0  1  0]
 [ 0  0  1  1  0  1  1  1  1  0  0 23  0  1  0  0]
 [ 0  0  2  0  0  0  2  1  0  0  0  1 22  1  1  0]
 [ 0  0  0  0  0  0  0  1  0  0  0  0  0 29  0  0]
 [ 0  0  0  0  0  0  1  0  1  0  0  0  1  0 27  0]
 [ 0  0  0  0  2  0  0  0  0  0  0  0  0  0  0 28]]
```

CONCLUSION:

Hence from the image classification algorithms we can observe that the CNN has the highest accuracy, and it was making good predictions and achieving best results. So, from the above we can state that.

CNN > SVM > RF

FOR OUR GIVEN DATA SET CNN IS THE BEST MODEL.

	SVM BASIC	SVM-PRETRAINED	RF BASIC	RF PRETRAINED	CNN	CNN-PRETRAINED
TEST ACC	44%	86%	38%	82%	48%	94%

REFERENCES:

1. https://www.analyticsvidhya.com/blog/2020/10/create-image-classification-model-python-keras/
2. https://keras.io/examples/vision/image_classification_from_scratch/#using-image-data-augmentation
3. https://www.kaggle.com/competitions/data-analytics-spring-2023-image-classification/discussion
4. https://github.com/krishnaik06/Machine-Learning-in-90-days