BioGecko

Vol 12 Issue 03 2023
ISSN NO: 2230-5807

IMAGE CLASSIFICATION USING CIFAR-10 DATASET

Vignesh.R[1], Prakash.V[2], Gokul Ajay.V.P[3], Siva.R[4], B.Selvapriya[5]*

Department Of Computer Science and Engineering, Bharath Institute of Science & Technology affiliated to Bharath Institute of Higher Education and Research, Chennai, Tamilnadu, India.

*Corresponding authors mail id: selvapriya.cse@bharathuniv.ac.in

ABSTRACT:

In this article, we undertake a comprehensive analysis of model scaling and find that striking a good balance between network depth, breadth, and resolution improves performance. On the basis of this finding, we suggest a novel way of scaling that adjusts depth, breadth, and resolution evenly using a basic yet highly effective compound coefficient. We provide proof that this approach is useful for expanding the scope of transfer learning. Further, we use neural architecture search to build a novel expand up a simple network in order to obtain several related models, known as transfer learning, that are both more accurate and efficient than prior ConvNets. In particular, our Efficient Net-B7 gets top-1 precision on ImageNet while being 8.4 x smaller and 6.1 x quicker on inference than the best current ConvNet. In addition to transferring well to CIFAR-10 and more3, we also obtain state-of-the-art precision on these other transfer learning datasets have proven effective in picture categorization and object recognition. In this article, we evaluate our deep learning model on the publicly available Cifar-10 dataset. In order to achieve high precision on the picture categorization job, numerous regularization techniques are combined with function optimization strategies like Adam and RMS.

Keywords: Efficient Net, Dimensions, Numerical Object, Neural Network, Matrices.

1. INTRODUCTION

Images are often classified and objects recognized using convolutional neural networks. Important characteristics are extracted by the convolution layer using filters or a feature extractor [1]. Several hyper parameters must be tweaked to get the most out of a deep neural network trained for image classification on the CIFAR-10 dataset. In an effort to prevent over fitting, we have experimented with dropout to simplify models and data supplementation to expand data size [2]. Convergence time can be drastically altered by making a deliberate decision about which function optimizer to use.

The network has utilized Adam as an enhancer. The network employs a ReLU activation function for all layers except the output layer, which employs a soft max activation function [3]. There are numeric identifiers in the data that need to be encoded as categorical. In training, the loss function is determined by comparing the predicted output to the observed output at regular intervals. Together, the Cross-entropy loss function and the Soft max activation at the end are used to improve the weights of the network [4]. To ensure the highest precision, this procedure is iterated for a predetermined amount of time, or "epochs."

The ability to identify objects through sight is crucial for human communication and engagement with the natural environment. Recognizing commonplace items like creatures, people, and foods requires little effort on our part, thanks to our superior visual identification skills [5]. Even when an object's location, size, pose, or lighting all change, a human brain can still easily recognizes it. Core object identification, which occurs in the human visual system's ventral stream, is a fundamental cognitive skill. It has been widely held that only living organisms possess the capacity for visual object identification. Numerous attempts at creating computer systems with capabilities similar

Vol 12 Issue 03 2023
ISSN NO: 2230-5807

to human object detection ability have been made in the area of computer vision. A human's sight identification abilities have been unmatched by machines despite decades of research [6].

CNNs for object identification follow the feed forward design of conventional neural networks in that it consists of several levels in a tiered fashion; both are influenced by organic neural systems. In particular, research has shown that the levels of a CNN correlate hierarchically with the layers of the human object identification system [7]. The effectiveness of deep neural networks for object identification is typically evaluated using standard databases in the field, such as ImageNet, CIFAR100, and CIFAR10. Non-human subjects have been used to rigorously evaluate these databases so that a full evaluation of human and AI visual capabilities can be made [8]. Even though some CNNs are said to be superior to humans at object identification, human performance is typically compared using data from and assessed using the ImageNet collection [9]. However, humans were not as effective as CNNs at recognizing the test pictures; only two people took part in the identification exercise, with one classifying 1500 images and the other 258 images out of a total of 100,000. In the meantime, CNNs are being taught to make predictions across the board in this test data collection. Giving humans the same test pictures as deep neural networks will enable for a more accurate contrast in object identification and, ultimately, will shed light on the relationship between organic and computer systems. However, if the number of object groups is too high, it becomes challenging to assess people's ability to recognize objects. To properly identify an item, for instance, it is difficult for a human to recall all of ImageNet's 1000 classifications [10].

Human identification errors can arise when people aren't conscious of the presence of the proper class, as suggested in. To evaluate the efficacy of deep neural networks and people in categorization, it may be useful to use a sample with a limited number of classifications. In, the writers use manipulated pictures to examine eight different types of objects. Work of a similar nature is performed with both five and ten classifications. While these experiments are useful for testing the efficacy of deep neural networks for visual identification, they do not take into account the most popular natural picture databases.

2. METHODOLOGY

2.1 Modules

- The CIFAR-10 Dataset
- Loading, wrangling, & splitting data preparation
- matplotlib to view images from dataset
- Transforming images into matrices of numbers (tensors)
- Training and Validation sets
- Detection

2.1.1 The CIFAR-10 Dataset

CIFAR-10, subset of the discontinued 80 Million Tiny Images (MTI) was one of the many possible datasets that was easily accessible within the framework of the PyTorch library (coming ‘plug-and-play’ in the .datasets module). I wish I could say there was a more romantic story behind it, but I chose CIFAR-10 rather arbitrarily.

Vol 12 Issue 03 2023
ISSN NO: 2230-5807

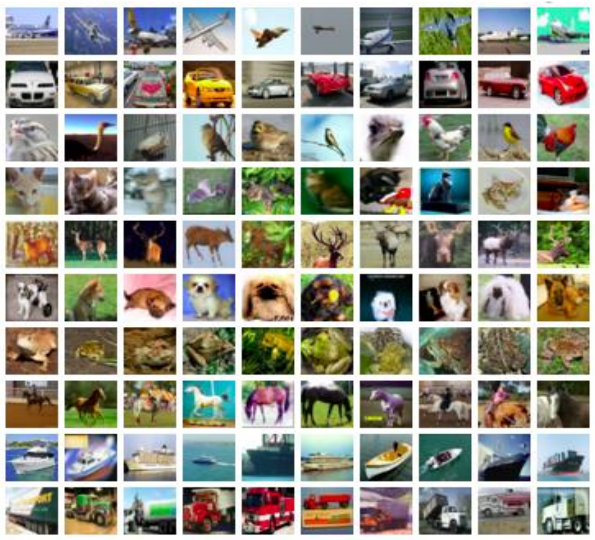


Figure 1: Sample Dataset

In figure 1, each row contains ten examples of the dataset's ten possible classes. Airplanes, Cars, Birds, Cats, Deer, Dogs, Frogs, Horses, Boats and Trucks. Then decided to use logistic regression to build a model that can guess which of 10 categories an input picture from the collection belongs to. Made up of 60,000 (50,000 for training/validation and 10,000 for test) 32x32 pixel images, CIFAR-10 has been an extraordinary resource for testing the accuracy of a variety of cutting-edge machine learning algorithms, as well as being a fantastic playground for machine learning experimentation.

2.1.2 Loading, wrangling, & splitting data preparation

It's an incredible convenience to be able to download the data directly from the PyTorch library, instead of the CIFAR-10 site. De-pickling files was a bit intimidating.

```
>>import torchvision# import the training
Set(first 50,000 imgs) by setting
train=True. >>trainset = torchvision.dataset.CIFAR10(root='./data',
train=True,download=True)# import the test set (10,000 imgs) by setting train=False>> testset =
torchvision.datasets.CIFAR10(root='./data', train=False, download=True)# define the ten labels manually.>> classes =
('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

Not much to explain mathematically or theoretically here. By accessing torchvision.datasets, we can access our CIFAR dataset. Indexing into data:

Let's access the first image in our dataset.

```
>>trainset[0]
(<PIL.Image.Image image mode=RGB size=32x32 at 0x7FF6E08F5908>, 6)
```

Indexing into trainset with typical notation returns two values. First, an intimidating string of letters and numbers, and 6. The fact that we are returned two values can create the correct image of trainset being a 50000x2 matrix, each

Vol 12 Issue 03 2023
ISSN NO: 2230-5807

example being a pair of image and correct answer, x and y. That intimidating first value is a PIL.image object a special code for that image encoded and saved using the Python Pillow library.

2.1.3 matplotlib to view images from dataset

The comparatively simple second term is the 'right answer' for that image, the label. Glancing at our predefined label array, we can see that 6 represent an image of a frog. But how do we view our little frog friend matplotlib to view images from dataset.

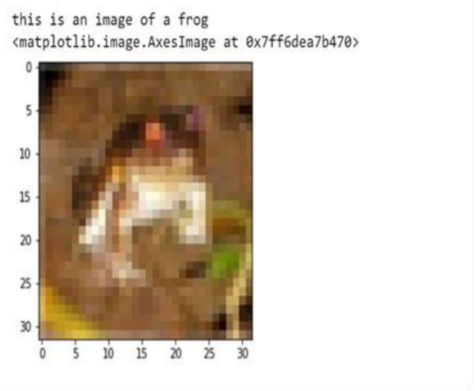
We can use the standard matplotlib.pyplot library to plot our images as colored pixel values.

```
# Depending on your integrated development environment (IDE), this enchanted procedure may be required.
>> %matplotlib inline
```

The above code is a magic function. Depending on your IDE, this code will prevent graphs created by matplotlib as shown in figure 2 to open in a different window (a popup) and will instead show them inline, underneath the code.
Now, we can plot our image.

```
>> example = 0# index into the 50,000 x 2 trainset vector, and store the label in the variable 'label'. Store the PIL object in variable 'image'.>> image, label = trainset[example] # print the label for the image by indexing label into classes.>> print("this is an image of a " + classes[label]# use imshow to show the PIL object image>> plt.imshow(image)
```

The image shown will show the training example from the training set. Executing this code will show us our first image from the training set, a frog, paired with its label.



```
this is an image of a frog
<matplotlib.image.AxesImage at 0x7ff6dea7b470>
```

Figure 2: Using matplot.lib to view images

2.1.4 Transforming images into matrices of numbers (tensors)

There isn't a way for an algorithm to truly *see* an image. That's why we need to find a way to represent an image with numbers, and individually feed each one of those numbers in to our algorithm as a feature. To understand how we do this, we must take a brief look at simple color theory. More specifically, the Red-Green-Blue (RGB) model.

Vol 12 Issue 03 2023
ISSN NO: 2230-5807

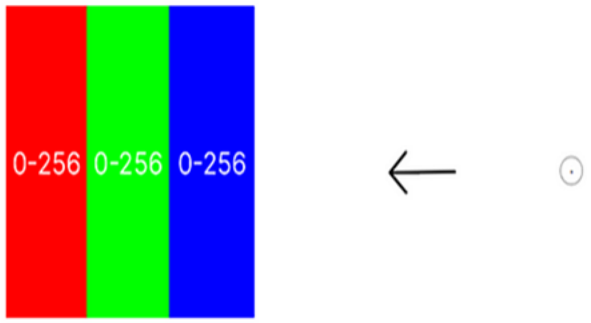


Figure 3: Intensity between 0 to 256

Every pixel on your screen, and every pixel in an image, is represented by three bars of primary red, green and blue. Each of these bars can be dialed up or down in intensity from 0–256 (why 256? think binary). Combinations of these three colors, each at some intensity between 0 and 256, create every color we know. Large blocks of color are made up of many pixels with the same RGB values. One pixel's color can be described by just three numbers.
Sensibly, then, we can represent a whole picture by defining it using its R, G, and B values for each pixel as shown in figure 3.

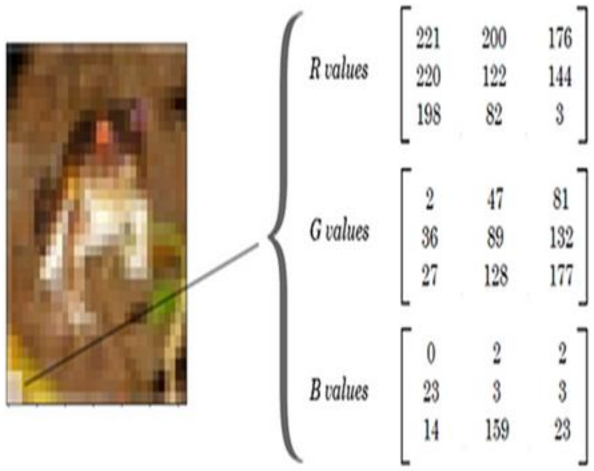


Figure 4: How to express a 3x3 square of pixels using matrices

In fig.4, formally we can describe an image using 3 matrices that hold the individual R, G, and B color channels of each pixel. Each of the three matrices will be the same dimensions, the *height* x *width* of the image in pixels. Thus, we can describe each image in our training set using a *3, 32, 32* matrix of pixel intensity values from 0–256. This gives us 3072 (3x32x32) *features* to plug into our dataset, if we unroll the image matrices into one long vector.

This is how an image sees. When an algorithm tries to predict the content of an image, it isn't looking the way humans tend to recall and identify the things we see. When an algorithm predicts the content of an image, it's really just comparing the number matrices with past number matrices and seeing which examples match up best.

Now we can confidently implement (the now trivial-seeming) code to do this image-to-matrix conversion.

```
>> import torchvision.transforms as transforms# directly access the training set again from .datasets, and set the transform keyword argument to transforms.ToTensor())>> datasetT = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transforms.ToTensor())
```

At this stage, it's most practical to think of tensors as *any numerical object* that can be operated on mathematically within a machine learning algorithm. Images are not tensors. Matrices, on the other hand, are indexing into our new, transformed training set; we see that the first value on each row (training example) is no longer a PIL.image object, but a tensor, and more specifically, a matrix of dimensions 3, 32, 32. This makes the dimensions of our total training set [50000, 3, 32, and 32].

```
>> example = imgTensor, label = datasetT[example]
>> print("size of image matrix: " + str(imgTensor.shape))
>> print("this is an image of a "+ classes[label]size of image matrix: torch.Size([3, 32,32])
this is an image of a frog
```

We can still view the image using *matplotlib.imshow* even if it's now in tensor format, requiring no change in syntax. As an added bonus, we can now index into custom values for the first dimension (x, 32, 32) to see the different color channels of the image individually (or :, all, to see the image normally)

```
# change the 0 (R) to 1 (G) or 2 (B) for different color channels. Additionally, recall that imgTensor represents just our first example.plt.imshow(imgTensor[0, 0:32, 0:32])
print("R channel of image 0 in dataset")
```

Note that *Python defaults to showing graphs in a green colormap*. Remember, when we use matplotlib to plot images, we're treating the images as *a heatmap of colored points* that form an image. If you want actual red channel values, pass in the tensor to .imshow and use

```
>> plt.imshow(imgTensor[0, 0:32, 0:32], cmap='Reds') # R
>> plt.imshow(imgTensor[1, 0:32, 0:32], cmap='Greens') # G
>> plt.imshow(imgTensor[2, 0:32, 0:32], cmap='Blues') # B
```

2.1.5 Training and Validation sets

Assume that if you're reading this article, you have a solid understanding about the significance for splitting your initial X set into training and validation sets. In essence:

- The training set is used to train the parameters of a given model, and usually takes up around 60% of a total dataset.
- The cross-validation set is used to evaluate the model during training (e.g., to see if gradient descent is working properly), to diagnose over and under fitting, and to adjust hyper parameters like the learning rate. It usually takes up around 20% of the total dataset.
- The test set is used to report the accuracy of the completed model. It usually takes up around 20% of the total dataset.

Since PyTorch already loads the test set independently, let's split our training set into an 80/20 ratio between training and cross-validation (CV) respectively. There are functions to do this that are built-in to PyTorch, but let's create a function to shuffle the indexes of the training set and to split the set into whatever ratio we want, from scratch.

```
#length of training examples matrix (how many training ex.)(m).>> m = 50000# percentage of training set dedicated to CV.>> pCV = 0.2# give the amount of cross-validation examples from m.>> mCV = int(m*pCV)# print the lengths>> print("amount of training examples: " + str(m - mCV))
>> print("amount of cross validation examples: " + str(mCV))amount of training examples: 40000
amount of cross validation examples: 10000
```

Now, we can define a function that will shuffle the indices of our original training set and allow us to split it up into the amounts described above.

```
>> def splitIndices(m, pCV):     >> """ randomly shuffle a training set's indices, then split
   indices into training and cross validation sets. Pass in 'm'
   length of training set, and 'pCV', the percentage of the
   training set you would like to dedicate to cross validation."""   # determine size of CV set.   >> mCV = int(m*pCV)
# create random permutation of 0 to m-1 - randomly shuffle all
   values from 0 to m.   >> indices = np.random.permutation(m)   # pick first mCV indices for training, and then
validation.   >> return indices[mCV:], indices[:mCV]
```

Basically what this code does is creates a random permutation of the numbers zero through 50000. By taking a random permutation on the number 50,000, all integers between 0 and 49,999 will be randomly shuffled into a list of 50,000 numbers. (E.g. a random permutation of 3 might result in list [2, 0, 1, and 3])

Then, we take the first mCV (amount of cross validation examples) items from the list, resulting in a list mCV long containing random indexes of the original training set. We then take the leftover 80% for the randomly shuffled indexes of the new training set.tl, dr; We randomly shuffled the indexes of the original training set, then took the first 10000 indexes as which examples from the original set will be going to the cross-validation set, and took the last 50000 as which examples will be going to our new training set.

2.1.6 Detection

- Object recognition is the process of accurately and quickly discovering pictures or movies that contain real-world things like people's features, flora, vehicles, etc.
- In order to identify every instance of a given object group, the object recognition method makes use of derived characteristics and learning methods.
- We begin by receiving a picture as input and then segmenting that image into smaller areas.
- Then, we'll treat each section as its own picture.
- Send all these picture areas to a CNN so it can assign labels to them.

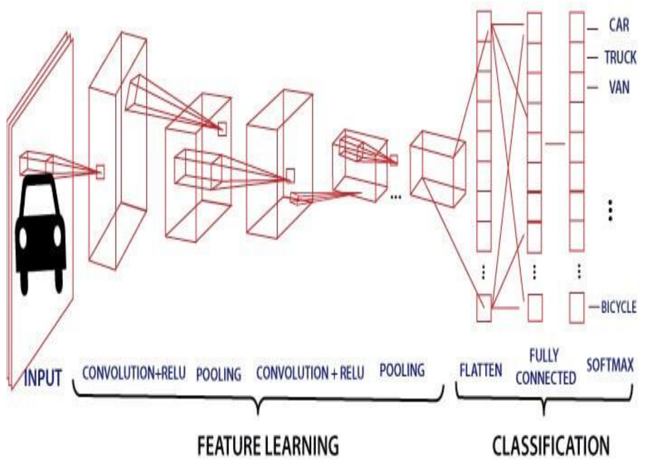


Figure 5: Classification Using CNN

Because convolutional neural networks are a type of neural network as seen in figure 5, they share all the other properties that make neural networks so useful. However, CNN's primary function is to perform picture processing. The specifics of their building design are as follows: it consists of two major sections.

The first unit in this type of neural network acts as a feature creator, setting it apart from others. This is accomplished via template matching via convolution filtration processes. First, the picture is filtered using multiple convolution kernels, yielding "feature maps," which are then adjusted (via an activation As we try different algorithms and refine the resulting features maps, we generate yet more features maps, which we then have to correct and reshape before starting the process all over again. The numbers of all the feature maps are summed up at the end to make a vector. This vector specifies the first-block output and second-block input.

The convolutional layer is the backbone of convolutional neural networks as shown in figure 6 and must always be the first layer. Its goal is to identify instances of a specific collection of characteristics within the raw pictures. Convolution filtering is used for this purpose; the basic idea is to "drag" a frame describing the feature onto the picture and then compute the convolution product between the feature and each pixel in the digitized image. It's then clear that a feature is synonymous with a filter.

Multiple pictures are fed into the convolutional layer, and the combination of each image with each filter is computed. The filters are a perfect match for the characteristics we're hoping to locate in the pictures. Each picture and filter combination yields a feature map indicating feature locations within the image, with greater values indicating stronger feature similarity between the associated feature location and the corresponding image pixel.

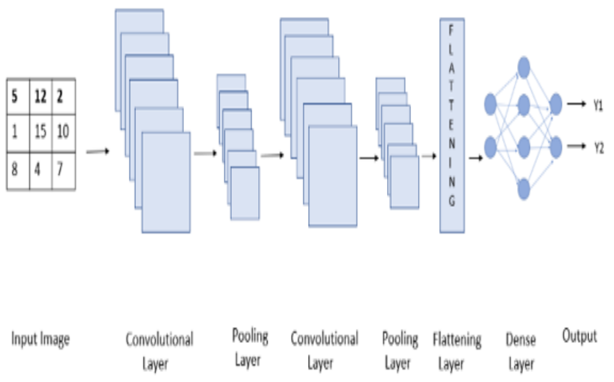


Figure 6: Layers in CNN Algorithm

The deep learning algorithm can autonomously derive important characteristics from visual data like pictures with the aid of a CNN, which consists of numerous convolution and pooling layers. CNNs learn a resilient order of characteristics that are constant under spatial, rotational, and translational transformations because of their multi-layered design. The above diagram illustrates the fundamental steps involved in building a CNN model. In tensor form, the numbers of individual pixels can depict any picture. The image's characteristics can be extracted with the aid of the convolution layers. (forms feature maps).

3. RESULT AND DISCUSSION

When it comes time to forecast a name, we feed the characteristics from the newly-captured pictures into our taught machine learning algorithm through the Output step. Recent deep neural networks have reportedly beaten humans on the CIFAR10 dataset for visual object identification, echoing findings on other datasets like ImageNet. There is still a need for further development and clarification of many remaining problems. Modern CNNs can outperform humans on tasks that humans find challenging, but they still can't do as well as humans on tasks that humans find simple. Approximately 80% of the test pictures used in CIFAR10 is ones that people can identify accurately with little to no training.

Human participants were found to classify these pictures very rapidly. That implies identification algorithms can still get better at visible object categorization, as neural networks are not as trustworthy as people. In that case, the current work's level-1 CIFAR10 simple subgroup may prove useful for gauging the efficacy of deep networks in the future. The fact that CNNs perform better than humans at object classification for challenging pictures but worse for simple ones suggests that we may still be far from confirming that deep neural networks can comprehend the meaning of scene, or at least understand images in the same way that humans do. It's likely that deep neural networks can identify a picture by drawing connections between it and others in their memory. Such a search for resemblance may be solely algorithmic and need not assume scene comprehension or extension. We assess the aforementioned deep neural networks on noise pictures to check their generalization abilities. All that's changed is the addition of zero-mean, zero-variance Gaussian noise to each of the 10,000 CIFAR10 test pictures.

CIFAR10's training pictures are used to train two networks, RNC and WRNC, which are then used to identify the noise images. To get a ballpark figure for how well humans would do on this novel dataset, we had a non-participating human participant identify 1000 noise pictures. (among 10000 and only with). This individual outperformed CNNs2 in terms of precision. Previous research on deteriorated pictures is consistent with our findings. It also supports the inference that fooling a deep neural network is relatively simple. When taught on chaotic pictures, deep neural networks are apt to improve their performance. In spite of this, we can also say that even the most advanced deep neural networks have not held true generalization capability, and that much more work needs to be done for them to achieve this human-like capability.

In order for neural networks to reliably identify objects, even when presented with challenging circumstances like aberrant hue or a chaotic background, they must be able to mimic the process directing scene comprehension. It is well-known that changes in location, posture, height, and lighting must be accounted for in any reliable visual identification system. These changes can be considered in an unrestricted fashion thanks to CIFAR10. The results of our trial with state-of-the-art CNNs on simple pictures (level 1) indicate that this feature can still be enhanced to meet the human level, despite the fact that some studies reveal the capacity of deep neural networks to learn invariance changes. The process of human visual identification may be better understood and neural networks improved through comparisons of CNN layers to reactions at the neuronal level in the human visual system, as in. Moreover, although all modern CNNs are feed forward, research suggests recurring networks may play a part in object identification. Perhaps future CNNs will take inspiration from this design to boost their efficiency. At the same time, research into interpretable deep neural networks is promising and deserves more focus.

Image categorization is a subfield of image processing that uses the innate characteristics of an image's subject matter to identify what it depicts. Image automated categorization technology has been applied to many different areas of development due to the increasing speed of technological advancement and the rising expectations of individuals in terms of the standard of their daily lives.

Traditional image classification methods have limitations when it comes to expressing the full range of features inherent in recognition objects due to the data's excessively high characteristic dimension, leading to less-than-ideal experimental results. In light of the foregoing, this article suggests a convolutional neural network–based approach to picture recognition.

In this exploratory method, deep learning and convolutional neural networks play a central role. In contrast to more conventional approaches to picture classification, both feature learning and classification can benefit from the deep convolution neural network model.

The Disadvantages Of Existing System

- Avoid encoding object positions and rotations, and expect to use a lot of training data.

- The challenge of communicating the issue to the network.
- A lack of positional invariance with respect to the incoming data.
- Huge amounts of data for training are necessary.

Proposed System

To improve CNN's generalization performance during the preliminary stages of training for image categorization, we suggest a new method of hierarchical transfer learning utilizing efficient net. Satisfactory generalization performance can be achieved with Transfer Learning using Efficient Net in a short amount of training time, which is crucial for real-time applications. To achieve Transfer Learning with Efficient Net, the suggested model can efficiently combine insights from multiple data sources. We use the CIFAR-10 and Image Net databases to verify the efficacy of our approach. The findings corroborate the hierarchy structure. Transfer Learning with Efficient Net greatly accelerates training, with an equivalent increase in precision for the early stages of learning in both the CIFAR-10 and Image Net cases.

Advantages of Proposed System

- It has very high precision in picture identification issues and can identify essential characteristics autonomously, without human intervention.
- It is easy to understand and fast to implement.
- Ability to develop an internal representation of a two-dimensional image.

4. CONCLUSION

In this study, we compared the performance of three distinct CNNs in terms of their ability to forecast outcomes on the CIFAR10 and CIFAR100 databases, which are very popular. We restricted our study to the most prevalent 10 groups from each cohort. Our main objective was to evaluate the accuracy of different networks by contrasting their results on the same datasets. We have provided a comprehensive forecast study to evaluate the networks' capabilities across a variety of object types. It's essential to remember that complicated frames frequently muddle the network's ability to identify and distinguish the image. There was also a discrepancy in accuracy rates between taught networks, despite the fact that in the actual world mattresses, sofas, and chairs are all distinct and readily recognizable items. The findings implied that taught networks incorporating transfer learning outperformed preexisting networks in terms of precision. It's the subject of a wide variety of studies and projects at the moment. It's adaptable and simple to incorporate into different environments, so it can be used in many contexts. One can still train the network and produce the desired model with only minimal requirements, even if the system requirements prevent the network from being taught on regular PC work.

REFERENCES

[1] Samson, C., Blanc-Féraud, L., Aubert, G., et al. (2000) A Level Set Model for Image Classification. International Journal of Computer Vision, 40:187-197.
[2] Alham, N.K., Li, M., Liu, Y., et al. (2013) A MapReduce-based distributed SVM ensemble for scalable image classification and annotation [J]. Computers & Mathematics with Applications, 66:1920-1934.
[3] Liu, Y. Guo, J., Lee, J. (2011) Halftone Image Classification Using LMS Algorithm and Naive Bayes [J]. IEEE Trans Image Process, 20:2837-2847.
[4] Lowe, D.G. (2004) Distinctive image features from scale—— invariant keypoints. International Journal of Computer Vision. 601-110.
[5] Moeskops, P., Viergever, M.A., Mendrik, A.M., et al. (2016) Automatic Segmentation of MR Brain Images With a Convolutional Neural Network. IEEE Transactions on Medical Imaging, 35:1252- 1261.

[6] “Kaggle competiton.” [Online]. Available: https://www.kaggle.com/c/cifar-10/leaderboard
[7] M. F. Aydogdu, V. Celik, and M. F. Demirci, “Comparison of three different cnn architectures for age classification,” pp. 372–377, 01 2017.
[8] M. F. Aydogdu and M. F. Demirci, “Age classification using an optimized cnn architecture,” in Proceedings of the International Conference on Compute and Data Analysis, ser. ICCDA ’17. New York, NY, USA: ACM, 2017, pp. 233–239. [Online]. Available: http://doi.acm.org/10.1145/3093241.3093277
[9] A. Krizhevsky, V. Nair, and G. Hinton, “Cifar-10 (canadian institute for advanced research).” [Online]. Available: http://www.cs.toronto.edu/ kriz/cifar.html
[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Advances in Neural Information Processing Systems 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824- imagenet-classification-with-deep-convolutional-neural-networks.pdf