CONVOLUTION REPORT

GROUP-9
(Arvind Chaurasia, Harika Beesetti)

Introduction:

The aim of this project is to develop a convolutional neural network (CNN) that can effectively differentiate between images of cats and dogs by identifying their unique features. To achieve this objective, we utilize a dataset sourced from Kaggle, which consists of 25,000 training images and 12,500 test images. Notably, the dataset maintains an equal distribution of images depicting cats and dogs.

Problem Statement:

The purpose of this assignment is to apply convolution networks(convnets) to image data set.

The Cats-vs-Dogs dataset is designed to classify images into either the dog or cat category.

Methodology:

Training from Scratch:

Initially, we started with a training sample of 1000 images, a validation sample of 500 images, and a test sample of 500 images.

Techniques employed to reduce overfitting and enhance performance included data augmentation, dropout regularization, and early stopping.

The network architecture consisted of multiple convolutional layers followed by max-pooling layers, dropout layers, and fully connected layers.

We trained the models using these techniques and evaluated its performance on the test dataset and training dataset.

Increasing Training Sample Size:

We increased the training sample size while keeping the validation and test samples the same. Network optimization was performed again from scratch.

Performance metrics were recorded and compared with the baseline results.

Optimizing Training Sample Size:

We adjusted the training sample size to achieve better performance than the previous steps.
The objective was to find the ideal training sample size for improved prediction results.

Using a Pretrained Network:

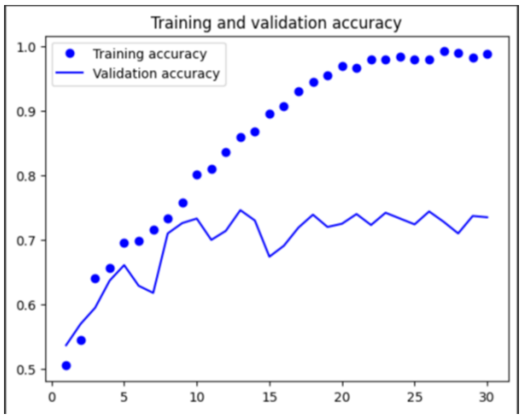
We repeated Steps 1-3 using a pretrained convnet. The sample sizes for training from scratch and using the pretrained network were varied to assess their impact on performance. Optimization techniques were applied to maximize model performance.

Insight and Observation:

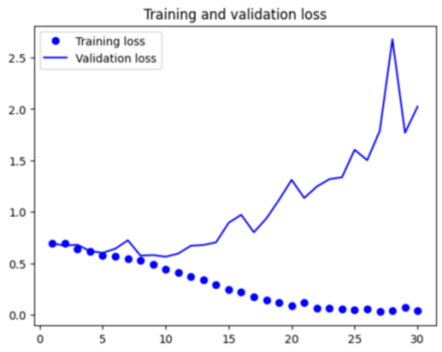
1. As we know in the first model, we were supposed to take the raining sample of 1000, a validation sample of 500, and a test sample of 500. At first model, we made a regular model with data normalization using 'rescaling' layer. We also used MaxPooling which helps in reducing the computational complexity of the model and controlling overfitting. We saw that the training accuracy increases over time when it reaches very high which is 99.2%, whereas the validation accuracy reaches its peak value at 72.5%. The validation loss continues to decrease as epochs keep on increasing, whereas the training loss keeps decreasing linearly as training proceeds.

   We got a test accuracy of 68.3%. (Due to the randomness of neural network initializations, we get numbers within one percentage point of that.) Because we have relatively few training samples, we have seen overfitting as it has a great training accuracy.

   We know that several techniques can help mitigate overfitting, such as dropout and weight decay (L2 regularization).



As you can see, the training accuracy is very high. Our model is overfitting, and the dataset is very small.



Again, in this graph, we can observe that the training loss is very less, and validation loss is increasing over epochs.

In order to combat overfitting, we used dropout method.

Here we got the train accuracy of 97.9% and validation accuracy of 73.5%, which is a bit improvement in our model. Further the test accuracy remained in the same range and had a value of 70.8%.

2. In the second question we increased the sample of the training data set. Initially, we had 1000 data samples in training, now we have 2000 datasets, and the validation and training sample sizes are the same.

   We worked with data augmentation, a specific computer vision technique and used almost universally when processing images with deep learning models.

As we know, overfitting is caused by having too few samples to learn from, rendering us unable to train a model that can generalize to new data.

In this model, we used 50 Epoch, we received a training accuracy of 90.3% and a validation accuracy of 82.1%. In this model, the test accuracy improved to 78.3%. As we have used the data augmentation.

Data augmentation takes the approach of generating more training data from existing training samples by augmenting the samples via several random transformations that yield believable-looking images.

The goal is that, at training time, our model will never see the same picture twice. This helps expose the model to more aspects of the data so it can generalize better.

3. In our assignment we also trained the model using data augmentation and dropout which is one of the regularization techniques used in neural networks to prevent overfitting and enhance generalization. We received a training accuracy of 88.7% and a validation accuracy of 74.3%. In this model, the test accuracy improved to 81.4%.

   The validation accuracy ends up consistently in the 70–85% range, a big improvement over our first model.

4. Leveraging a pre-trained model

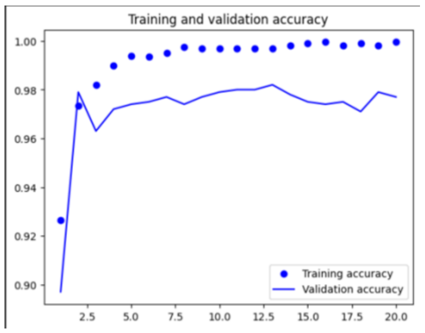
   In 4th step, we have trained three models.

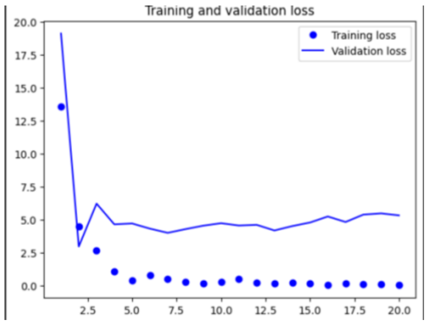
   1st model: Training Accuracy: 96.7% validation accuracy: 98.4% and Testing accuracy: 78.5%

   After fine tunning
   1st model: Training Accuracy: 98.2% validation accuracy: 97.9% and Testing accuracy: 97.6%

   2nd model: Training Accuracy: 99.4% validation accuracy: 98.5% and Testing accuracy: 97.9%

   3rd model: Training Accuracy: 99.5% validation accuracy: 97.8% and Testing accuracy: 97.9%





As we can see the fine tuning helped to improve the training accuracy for the first model. A common and highly effective approach to deep learning on small image datasets is to use a pre-trained model. A pre-trained model is a model that was previously trained on a large dataset, typically on a large-scale image-classification task. We used the VGG16 architecture.

There are two ways to use a pre-trained model: feature extraction and fine-tuning. We covered both. Feature extraction consists of using the representations learned by a previously trained model to extract interesting features from new samples.

These features are then run through a new classifier, which is trained from scratch. Fine-tuning consists of unfreezing a few of the top layers of a frozen model base used for feature extraction, and jointly training both the newly added part of the model (in this case, the fully connected classifier) and these top layers.

<u>Conclusion:</u>

We got a good testing accuracy score of with just using small set of data from a large image data set. Here's how we made sure our model didn't overfit:

- Since we couldn't get more training data, we used a trick called data augmentation to make the most of what we had.
- We kept our model simple by not letting it have too many parameters.

- We also made sure the weights in our model didn't get too big, which helped keep things simple.
- During training, we randomly turned off some parts of our model to prevent it from getting too good at just memorizing the training data.

We tried different setups with the size of our training and validation sets, and we also compared models we built from scratch with ones that were already partly trained. Turns out, having more training data or adjusting the validation set size made our model better. But when we tried adding more data through data augmentation, it didn't really help.

Overall, using pretrained models worked better when we didn't have a lot of training data.