**Convolution Neural Networks**

**Introduction:**

Convolution Neural Networks are defined as a class of deep learning neural networks primarily used in Image Recognition, Object Detection, and Image Classification. Basically, it functions as a filter applying filters to the input image. An image is convolution when a set of filters is slid over the image, resulting in several feature maps showing the image's various aspects.

**Summary:**

This dataset consists of 25,000 images featuring cats and dogs. For predicting test images, these images are fed into a convolutional neural network. In this report, each Convolutional Neural Network (CNN) model is evaluated with different training sample sizes. Afterwards, CNN models with different training sample sizes will be analyzed for their performance. Besides the different sample sizes, augmented data is used to build CNN models from scratch and pretrained models.

The picture shows the architecture of the model:

A screenshot of a computer program

Description automatically generated

Observations:

As the model is trained from scratch, random weights are assigned to it and gradually updated throughout. To achieve good performance, labelled data must be of good quality and large. Based on this dataset, the initial model built using Training from Scratch was 0.717 accurate and 0.566 validated.

A Pretrained neural network model trained on a large dataset is used as a starting point for a similar and smaller dataset. This improves performance, reduces overfitting, saves time and computational resources. The initial model built on this dataset using a pretrained neural network achieved 0.971 accuracy and 5.618 validation loss.

**Model adding Data Augmentation:**

When the model added with data augmentation the test accuracy has increased from 0.717 to 0.826, and the validation loss has decreased from 0.566 to 0.493 Meanwhile, in the pretrained model the accuracy also increases to 0.979 and validation loss has decreases to 1.628. This is because data augmentation is a technique used in model training to apply variations like rotating, flipping, and zooming to each image. This creates new forms of images and helps to learn more robust features and improve performance while reducing overfitting, causing an increase in validation loss in pre-trained networks.

**Sample Increment for Training model:**

A larger training sample in a CNN improves model performance by allowing it to learn more and capture more patterns. More samples also reduce overfitting, but low- quality data can increase it. More training data improves the model’s performance, but there’s a limit. After a certain point, additional data doesn’t provide much useful information because the model has already learned all the important features, causing its performance to stabilize.

After increasing the data with 3500 the accuracy decreases from 71.7% to 67.4% with a validation loss of 0.606. Then again, the data increases to 4500, the accuracy increases to 72.8% with a validation loss of 0.593. After that, the data again increases to 5500 which is the best sample size for network model having accuracy of 74.3% and validation loss of 0.578.

Multiple CNNs were built by using pre-trained models and varying the training sample size. Table 2 shows the test results of each model. The first model, which used 1000 samples, achieved a 97% test accuracy with validation loss of 5.618. The next model, using 2000 samples with data augmentation, improved the test accuracy to 97.9% and reduced the validation loss to 1.628. Another model with 3500 samples showed a test accuracy of 97.4% but had a higher validation loss of 5.38. The model with 4500 samples achieved a test accuracy of 97.3% and a validation loss of 3.625. Lastly, the model with 5500 training samples maintained a high-test accuracy of 97.0% and achieved a higher validation loss of 6.264.

Table 1: CNN’s built using training from scratch.

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Training from scratch | | |
| Performance Metric | Accuracy | Validation Loss | No. of epochs |
| Initial model  (Training=1000, Validation=500, Test=1000) | 0.717 | 0.5667 | 30 |
| Data Augmentation  (Training=2000, Validation=500, Test=1000) | 0.826 | 0.493 | 100 |
| Increased Training Data  (Training=3500, Validation=500, Test=1000) | 0.674 | 0.606 | 30 |
| Increased Training Data  (Training=4500, Validation=500, Test=1000) | 0.728 | 0.593 | 30 |
| Optimal Training Data  (Training=5500, Validation=500, Test=1000) | 0.743 | 0.578 | 30 |

Table 2: CNN’s using Pretrained Model.

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Pretrained Model | | |
| Performance Metric | Accuracy | Validation Loss | No. of epochs |
| Initial model  (Training=1000, Validation=500, Test=1000) | 0.971 | 5.618 | 30 |
| Data Augmentation  (Training=2000, Validation=500, Test=1000) | 0.979 | 1.628 | 60 |
| Increased Training Data  (Training=3500, Validation=500, Test=1000) | 0.974 | 5.388 | 30 |
| Increased Training Data  (Training=4500, Validation=500, Test=1000) | 0.973 | 3.625 | 30 |
| Optimal Training Data  (Training=5500, Validation=500, Test=1000) | 0.970 | 6.264 | 30 |

**Conclusion:**

In conclusion, CNN trained from scratch gave moderate accuracy (71.7%) and low validation loss. Pretrained models resulted in significant performance boost with impressive accuracy (97.1%), but higher validation loss, showcasing the power of transfer learning.

Data augmentation significantly improved accuracy and reduced overfitting in both scenarios. Increasing training sample size generally improved performance, with a noted trade-off between data size and overfitting observed in the pretrained model. Based on our analysis, we have found that the ideal number of training data samples for both cases is approximately 5500.

To conclude, using pre-trained models is a successful approach for image classification, incorporating data augmentation techniques can be advantageous, and it is essential to carefully evaluate the size of the training data to achieve optimal outcomes.

 A graph with a line

Description automatically generated