**ASSIGNMENT-2 CONVOLUTION – SUMMARY REPORT**

1. **Consider the Cats & Dogs example. Start initially with a training sample of 1000, a  
   validation sample of 500, and a test sample of 500 (like in the text). Use any technique  
   to reduce overfitting and improve performance in developing a network that you train  
   from scratch. What performance did you achieve?**

**Initializing the Training Dataset:**

A DataFrame named `train` is created to store the training data. The `load\_dataset` function processes image files from the `TRAIN\_DIR` directory, assigning labels to each image (e.g., 0 for cats and 1 for dogs). These images and their labels are then added to the `train` DataFrame. To prevent bias from any inherent ordering in the data, the dataset is randomized using the `sample` function, and the DataFrame index is reset for consistency. A preview of the first few rows of the `train` DataFrame is provided to review the dataset's composition.

**Loading Datasets for Testing and Validation:**

Similarly, datasets for testing and validation are prepared by loading and labeling image data into corresponding DataFrames.

**Data Visualization and Exploration:**

This section focuses on examining and visualizing the training, testing, and validation datasets. A count plot is generated to show the distribution of labels within the dataset, offering insights into the class balance between cats and dogs. Additionally, the total number of images in the training dataset is calculated and displayed. This exploratory step is essential for understanding the dataset's structure, which is crucial for the model design and training phase.

**Visualizing Images in a Grid:**

**Library Imports:** The process begins with importing necessary libraries, including Keras for handling image operations and Matplotlib for visualizing images.

**Setting Up the Visualization Canvas:**

A Matplotlib figure, measuring 20x20 inches, is set up as the canvas for displaying a collection of images in a structured grid format.

**Choosing a Subset from the Training Data:**

A specific portion of the training dataset, particularly the first 25 entries from the `train` DataFrame, is selected for visualization. These images are arranged into a 5x5 grid.

**Looping Through and Visualizing Images:**

Each image's path and label are retrieved from the `train` DataFrame, and for every image:

- A subplot within the 5x5 grid is designated.

- Images are loaded as NumPy arrays using Keras' `load\_img` function.

- Each image is displayed on its subplot with its label as the title.

- Axis labels and ticks are removed for better visual clarity.

**Rendering the Image Grid:**

After iterating through the selected images, the compiled grid is displayed within the Matplotlib figure.

**Convolutional Neural Network (CNN) Model for Image Classification Using Keras:**

**Model Architecture Definition:**

A sequential model named `model\_1` is defined, consisting of several layers including convolutional layers, max-pooling layers, a flatten layer, and fully connected layers.

**Convolutional Layers:**

Multiple convolutional layers are added to extract features from input images, starting with a layer that has 16 filters, a 3x3 kernel size, and ReLU activation. Subsequent layers have 32 and 64 filters.

**Max-Pooling Layers:**

Max-pooling layers are inserted after certain convolutional layers to downsample the feature maps.

**Flatten Layer:**

A flatten layer is included to convert multi-dimensional feature maps into a one-dimensional vector.

**Fully Connected Layers:**

A dense layer with 2 output units and sigmoid activation is added for binary classification.

**Model Compilation:**

The model is compiled with categorical cross-entropy as the loss function, Adam as the optimizer, and accuracy as the evaluation metric.

**Model Training:**

The model is trained using the training dataset for a specified number of epochs, with the entire training dataset used in each epoch. Validation data is used to evaluate the model's performance.

**Data Augmentation and Preprocessing for the Training Set:**

**Library Importation:**

Keras' `ImageDataGenerator` is imported for image preprocessing and augmentation.

**Setting Up ImageDataGenerator:**

An instance of `ImageDataGenerator` is created with data augmentation techniques such as rescaling, shearing, zooming, and horizontal flipping.

**Initializing the Training Data Generator:**

A generator for the training data is established using `train\_datagen` with `flow\_from\_dataframe`, specifying image paths and labels from the `train` DataFrame, target image size, batch size, and class mode.

**Building a CNN Model for Image Classification with Keras:**

**Model Foundation:**

A Sequential model is instantiated for stacking layers linearly.

**Incorporating Convolutional Layers:**

The model includes a Conv2D layer with 32 filters, a 3x3 kernel size, and ReLU activation, followed by a MaxPooling2D layer and additional convolutional and max-pooling layers.

**Flattening Output:**

A Flatten layer transitions the model from convolutional layers to fully connected layers.

**Adding Dense Layers:**

A dense layer with 64 neurons and ReLU activation is included for learning complex patterns, followed by a final dense layer with 2 neurons and sigmoid activation for binary classification.

**Model Evaluation:**

The CNN model, defined, compiled, and trained previously, is evaluated using the testing dataset to assess its performance.



**2. Increase your training sample size. You may pick any amount. Keep the validation and test samples the same as above. Optimize your network (again training from scratch). What performance did you achieve?**

The training dataset has been expanded to 5000 images to enhance the model's accuracy.

**1. Importing Libraries:**

The process begins with importing essential libraries:

- Pandas for data manipulation.

- Seaborn for data visualization.

- Matplotlib for plotting.

**2. Creating a Training DataFrame (train\_v2):**

- A new Pandas DataFrame called `train\_v2` is created to store the training data.

- Image data and their corresponding labels are loaded from a directory specified by `TRAIN\_V2\_DIR` using the `load\_dataset` function.

- The `load\_dataset` function reads the image files, processes them, and assigns labels to each image.

**3. Shuffling the Dataset:**

- The `train\_v2` DataFrame is shuffled to randomize the order of the data, preventing biases from a specific data order.

- Shuffling is done using the `sample` function with `frac=1` to keep all data and reset the index.

**4. Creating a Count Plot of Labels:**

- A count plot of the labels in the `train\_v2` DataFrame is generated with Seaborn.

- The `countplot` function visualizes the distribution of labels in the dataset.

- The plot is displayed in a 10x6 figure, titled "Label Count Plot."

- The x-axis is labeled "Label," and the y-axis is labeled "Count."

**5. Printing Total Image Count:**

- The total number of images in the `train\_v2` dataset is calculated and printed.

- This is done by determining the length of the `train\_v2` DataFrame.

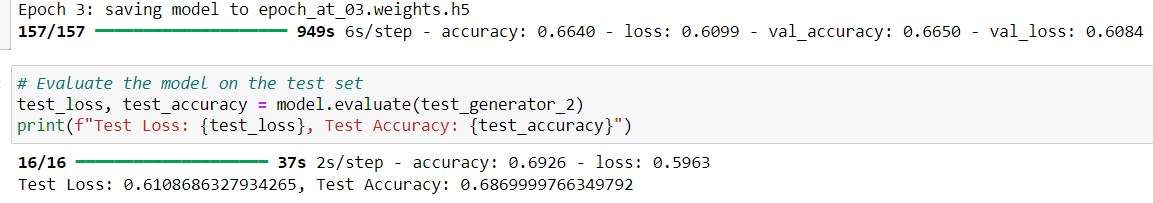
**6. Displaying the Plot:**

- The plot is displayed using `plt.show()`.

**Repeating Steps for Model Classification Using Convolutional Neural Network (CNN):**

The same procedures are repeated for model classification using a Convolutional Neural Network (CNN).We repeat the same steps using the above question for Model classification Using Convolutional Neural Network(CNN)

Performance achieved from the new Dataset:



**3. Now change your training sample so that you achieve better performance than those from Steps 1 and 2. This sample size may be larger, or smaller than those in the previous steps. The objective is to find the ideal training sample size to get best prediction results.**

Based on the first and second question, it can be inferred that increasing the dataset size can improve the model's accuracy.

**Enhanced Dataset:**

The training dataset has now been increased to 22,000 images. Following the steps outlined previously, the model was trained with this expanded dataset.

**Observations:**

After training with 22,000 images, it was observed that the accuracy improved.

Results:

The results after training with 22,000 images can be seen below:

**Model Evaluation:**

A screenshot of a computer

Description automatically generated

A graph with lines and numbers

Description automatically generated

**4. Repeat Steps 1-3, but now using a pretrained network. The sample sizes you use in Steps 2 and 3 for the pretrained network may be the same or different from those using the network where you trained from scratch. Again, use any and all optimization techniques to get best performance**

**MobileVnet** is the pretrained model used here to train the dataset.

Model Architecture

**Base Model:**

The model's foundation is MobileNetV2, pre-trained on the comprehensive ImageNet dataset. This ensures the model benefits from high-quality feature representations for a wide variety of images. The pre-trained model is adapted to an input shape of 150x150 pixels with three color channels (RGB), meeting the specific requirements of the task.

**Model Customization:**

To tailor MobileNetV2 for binary classification, the network's top layer (originally designed for 1000 classes) is removed. This modification allows for the addition of custom layers suited to the task.

**Global Average Pooling:**

After the base model, a Global Average Pooling 2D layer is applied. This layer reduces the dimensions of the feature maps to a single vector per map, significantly lowering the model's complexity and computational cost while preserving essential spatial information.

Dense Layers: The flattened output is then passed through a Dense layer with 1024 neurons, employing the ReLU activation function for non-linear processing. This layer serves to learn high-level features derived from the base model's output. The final Dense layer, consisting of a single neuron with a sigmoid activation function, produces the binary output indicating the class probabilities (cat or dog).

**Model Compilation**

Optimizer:

The RMSprop optimizer is employed with a learning rate of 0.001. This optimizer is chosen for its effectiveness in managing gradients during mini-batch learning, which is especially useful for tasks with complex, non-linear objectives such as image classification.

Loss Function:

Binary crossentropy is used as the loss function, ideal for binary classification tasks. It aims to minimize the difference between the actual and predicted probability distributions.

Metrics:

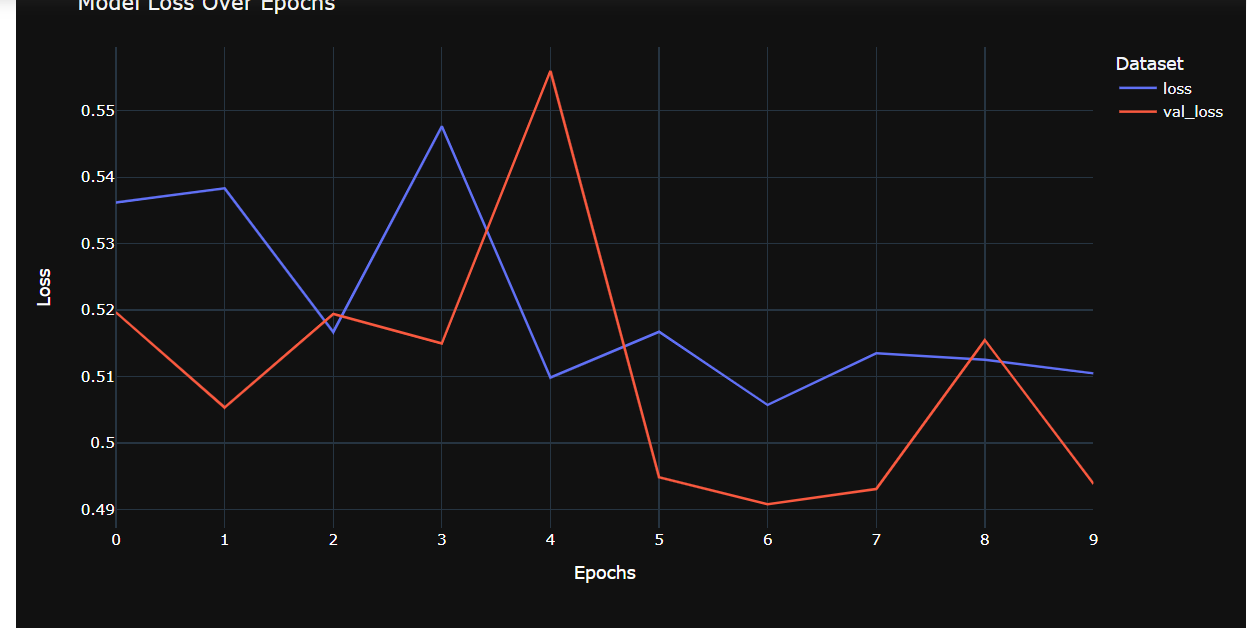
Accuracy is utilized as the performance metric, offering a clear measure of the model's capability in classifying images as either cats or dogs.

A screenshot of a graph

Description automatically generated

A graph on a black background

Description automatically generated



**CNN model on the training dataset for 10 epochs while validating its performance on a separate validation dataset**

model\_pre.fit(...): This method is used to train the CNN model (model\_pre) with the specified training and validation datasets. It performs the following tasks:

- training\_v3\_set: The dataset used for training the model.

- steps\_per\_epoch=len(training\_v3\_set): Sets the number of training steps per epoch to the total number of batches in the training dataset, ensuring the entire dataset is utilized each epoch.

- epochs=10: The model undergoes training for 10 epochs, meaning the entire training dataset is passed through the model 10 times.

- validation\_data=validation\_set: The model's performance is assessed on the validation dataset after each epoch, enabling evaluation of its ability to generalize to unseen data.

- validation\_steps=len(validation\_set): Sets the number of validation steps to the total number of batches in the validation dataset.

**Evaluation of the model based on the Training dataset:**

- model\_pre.evaluate(testing\_set): This code line calls the evaluate method on the model\_pre model to gauge its performance using the testing dataset. The evaluation procedure involves:

- testing\_set: This dataset serves as the input for evaluation, comprising data instances and their corresponding labels.

- The model processes the testing dataset, computes performance metrics such as accuracy and loss, and provides these metrics as evaluation results.

- The evaluation outcomes are typically returned as a tuple of values, which commonly include accuracy, loss, and other specified metrics from the model compilation. These results are pivotal for assessing the model's ability to generalize to new data and for making informed decisions regarding its effectiveness.

**Visualization of Model Training Outcomes:**

**Retrieving Training Metrics:**

Initially, data is extracted from the history\_pre object, which contains recorded metrics from the model's training phase, encompassing both accuracy and loss metrics for training and validation.

**Plotting the Accuracy:**

An accuracy chart is generated to visualize the evolution of training accuracy (shown in blue) and validation accuracy (depicted in red) across epochs. The horizontal axis represents the epoch count, while the vertical axis denotes accuracy percentage. Titled 'Accuracy Graph', the chart includes a legend distinguishing between training and validation accuracy.

**Illustrating the Loss:**

Similarly, a loss graph is plotted to compare training loss (in blue) with validation loss (in red) over the epochs. The horizontal axis indicates epochs, and the vertical axis displays loss values. Named 'Loss Graph', this graph is accompanied by a legend distinguishing between training and validation loss.

**Graph Display:**

Both the accuracy and loss charts are displayed using plt.show() for visual examination.

Overview:

Enhancements in CNN Performance Through Expanded Datasets:

**Diverse Data Augmentation:**

Increasing the dataset size introduces a wider range of examples, enhancing the model's capacity to generalize effectively to new data. A larger image corpus exposes the model to diverse conditions, fostering robust learning.

**Mitigating Overfitting Risks:**

Larger datasets mitigate the risk of overfitting by expanding the scope of learning with a comprehensive set of examples. This breadth of data encourages the model to focus on learning significant patterns rather than memorizing specific instances.

**Improved Feature Detection:**

With more data available, a CNN can refine its ability to detect subtle features, leading to a deeper understanding of the data. This capability enables the recognition of intricate and abstract features, capturing complex data relationships effectively.

**Optimization Advantages:**

Expanded datasets contribute to a more stable and accurate optimization process, facilitating the model's convergence towards optimal solutions. Enhanced stability in gradient updates promotes smoother optimization trajectories.

Strategies for Efficient Training on Large Datasets:

**Batch Normalization:**

Normalization of layer inputs through batch normalization enhances training stability and promotes model generalization.

**Adaptive Learning Rate Adjustments:**

Implementing schedules for adjusting learning rates can significantly aid in model convergence, particularly crucial when handling extensive datasets.

**Data Augmentation Techniques:**

Artificially expanding datasets through transformations improves model generalization, especially beneficial for large datasets.

**Regularization Methods:**

Incorporating regularization techniques such as dropout and L2 regularization helps the model focus on learning generalized patterns.

**Ensemble Techniques:**

Utilizing ensemble methods that combine predictions from multiple models can leverage diversity to enhance overall performance.

**Transfer Learning Application:**

Leveraging pretrained models, which have learned feature representations from extensive datasets, can yield superior performance for specific tasks.