**Report**

**Introduction:**

A field of deep learning models called Convolutional Neural Networks (CNNs) is made to handle data that has a topology resembling a grid, such an image that is represented as a 2D grid of pixels. The term "convolutional" denotes that the network makes use of the convolutional mathematical operation, a particular sort of linear operation. CNNs are made to learn spatial feature hierarchies from tasks that need the input of pictures in an autonomous and adaptable manner.

Numerous tasks involving text and picture recognition have proven effective for CNNs. They are made up of many layers of tiny neuronal groupings that analyze what are known as receptive fields—a piece of the incoming picture. The layers that pool the output of the neuron collections and feed it into the subsequent layer are sandwiched between these layers. Next, classification tasks are performed using the CNN's final output.

**Dataset:**

Images of dogs and cats make up the standard computer vision dataset known as Dogs and Cats. There are 25,000 photos in the collection, equally divided between 12,500 photos of dogs and 12,500 photos of cats. These pictures are used in binary classification tasks, which train a model to identify between a cat and a dog in a given image. The dataset's vastness and relative simplicity make it a popular choice for machine learning and deep learning courses and benchmarks, even if it remains sufficiently difficult to train on. If you're unfamiliar with CNNs and picture classification, this is an excellent dataset to start with.

**Data Preprocessing:**

First the data is loaded in the R environment then the zip file is extracted and check the number of files that contain the images of cats and dogs in separate folder.

"PetImages" is the base directory where the photos are kept.

To normalize the pixel values in the photos, an image\_data\_generator() is made with a rescale factor of 1/255. Additionally, 10% of the data will be utilized for validation, according to the validation split of 0.1 that is chosen.

A training dataset is created using the flow\_images\_from\_directory() function. The function accepts the following parameters: the class mode, the path to the directory, the data generator, the target picture size, the batch size, the subset type, and a reproducibility seed. It locates 22,500 photos in the given directory, divided into the dog and cat groups.

Similarly, the flow\_images\_from\_directory() method is used to produce a validation dataset. For validation, it locates the remaining photos in the directory.

**Model Building:**

Model is built by specifying the number of layers that takes the photo of animal as input and then generates the output after processing to the next layer. Following are the number of layers:

The model consists of 17 layers in total. Here’s the breakdown:

1. **4 Convolutional layers (layer\_conv\_2d)**

The model has four Conv2D layers. These convolutional layers are going to learn the filters, also called kernels, which are manually-engineered in conventional CNNs. Using the kernel filter, each filter modifies a portion of the picture (specified by the kernel size). The weights of the kernel and the biases dictate the transformation that is used. The output shape illustrates how the layer will change the image.

1. **4 Max Pooling layers (layer\_max\_pooling\_2d)**

The model has four MaxPooling2D layers. These layers assist the model better generalize while also reducing computation by reducing the spatial dimensions (height, breadth) of your input.

1. **1 Flatten layer (layer\_flatten)**

The model has one flatten layer. The input is flattened by this layer. It has no bearing on the batch size. When moving from convolutional or pooling layers to dense layers, this technique is employed.

1. **3 Dense layers (layer\_dense)**

Your model has three dense layers. Each node in the layer is connected to every other node in the layer above, making them completely connected layers.

1. **3 Batch Normalization layers (layer\_batch\_normalization)**

In your model, there are three BatchNormalization layers. These layers apply a modification that keeps the activation standard deviation near to 1 and the mean activation close to 0, normalizing the activations of the preceding layer at each batch. It offers some regularization, noise robustness, and learning acceleration.

1. **2 Dropout layers (layer\_dropout)**

Your model has two dropout layers. To assist avoid overfitting, these layers randomly change the fraction rate of input units to 0 at each update during the training phase.

1. **Output Layer:** The likelihood that the image depicts a dog (or cat) is output by the single unit in the final Dense layer with a sigmoid activation function.

* Max pooling Layers

The purpose of this layer is to decrease the input volume's width and height so that the subsequent convolutional layer may use it. It uses the MAX operation to spatially enlarge each depth slice of the input individually. The most popular kind is a pooling layer that has 2x2 filters applied with a stride of 2. This filtering method discards 75% of the activations by down-sampling each depth slice in the input by 2 in both width and height.

* ReLU Function

In deep learning models, the most often employed activation function is the Rectified Linear Unit (ReLU). If the function gets any negative input, it returns 0; otherwise, it returns the value that it received, x, if it is positive. The formula for it is therefore

f(x) = max(0, x)

Since most real-world input is non-linear and we want neurons to learn these non-linear representations, the activation function adds non-linearity to a neuron's output, which is crucial.

Your model has 3,902,529 total parameters, of which 3,899,457 can be trained and 3,072 cannot. Because the running means and variances in the BatchNormalization levels are not changed during training, these layers provide the non-trainable parameters. 14.89 MB is the estimated size of the model. The amount of computing needed for training and inference might vary depending on the size and number of parameters in the model; larger models require more processing power.

**Compilation of the model:**

Using the binary crossentropy loss function, the model will learn to classify pictures as either a dog or a cat. It will then use the Adam optimization technique to update its weights and assess its performance based on accuracy. Fitting the model to the data using the fit function would be the next step after this. The model will begin to learn from the training data as a result, initiating the training phase. The model may be used to forecast fresh, unobserved data once it has been trained.

**Model Fitting:**

**Fit\_generator:** This function trains the model over a predetermined number of epochs, or dataset iterations. Because it makes use of Python generators, you can loop through data endlessly. When you have a big dataset that won't fit in memory, this is quite helpful.

**train\_generator:** The model will learn from this set of training data. In the code, it was defined earlier.

**steps\_per\_epoch:** This is the number of sample batches that must be processed before concluding one epoch and initiating the subsequent one. It is set to 10 here.

The number of times the learning algorithm will run over the complete training dataset is known as an epoch. It is set to 10 here.

The output displays the model's accuracy and loss following the last epoch. The model's performance on the training set of data is shown by the loss. The better the model matches the actual data with its predictions, the less the loss. The frequency with which the model's predictions come true is indicated by its accuracy. After the last epoch, the model's accuracy on the training set is 63.75% in this case.

**Testing the Model:**

The image\_load() method loads the image from the given location. To match the input shape that the model anticipates, the picture is downsized to 200x200 pixels.

The image\_to\_array() method is then used to transform the image into an array. The model requires input in the form of an array; hence this is required.

The array\_reshape() function is used to provide the picture array an additional dimension. This is because, even if we're just guessing on one image, the input still has to be in a batch format, since the model requires it.

The model makes a prediction on the test image using the predict function. The output of this function is a number between 0 and 1, thanks to the sigmoid activation function in the last layer of the model.

If the output is greater than or equal to 0.5, it prints “Cat”, otherwise it prints “Dog”. This is based on how the labels were encoded during the training phase (assuming 0 is for dogs and 1 is for cats).

**Conclusion**

While CNNs can be computationally intensive due to the high dimensionality of image data, techniques like pooling are used to manage this complexity. Pooling helps to reduce the computational cost, memory usage, and also adds robustness to the model. It’s a trade-off between computational efficiency and model performance. The key is to design the network architecture such that it is complex enough to capture the necessary features for the task at hand, but not so complex that it becomes computationally prohibitive to train.