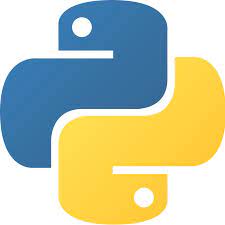
Cats and Dogs Image Classification Using Deep Learning with Python

The Ultimate Guide for Building Convolutional Neural Networks (CNN) to Classify Images of Dogs and Cats using Keras and TensorFlow.





**Blog By: Chaithanya Vamshi Sai**

**Student Id: 21152797**

# **Introduction**

In todays’ data industry, there’s a strong belief that when it comes to working with unstructured data, especially image data, deep learning models are the best. Deep learning algorithms like Convolutional Neural Networks (CNN) undoubtedly perform extremely well on image data.

Deep Learning algorithms can handle large amounts of data but it requires a large amount of data to train and also requires a lot of computing resources.

If we provide the right data and features, Deep learning models can perform adequately well and can even be used as an automated solution.

In this blog, I will demonstrate and show how we can harness the power of Deep Learning and perform image classification using Convolutional Neural Networks (CNN) to Classify Images of Dogs and Cats using Keras and TensorFlow.

# **Steps to Build an Optimised Convolutional Neural Network Image Classifier Model using Keras, TensorFlow & Hyperparameter Tuning with Keras Tuner.**

1. Problem Statement
2. Importing Libraries
3. Unzipping and loading the dataset
4. Data Exploration
5. Data Augmentation
6. Model Building
7. Model Evaluation: CNN Model Loss and Accuracy Results
8. Hyperparameter Tuning of the CNN Model using Keras Tuner
9. Model Building: CNN Model with the Best Hyperparameters
10. Model Evaluation: Optimised CNN Model Loss and Accuracy Results

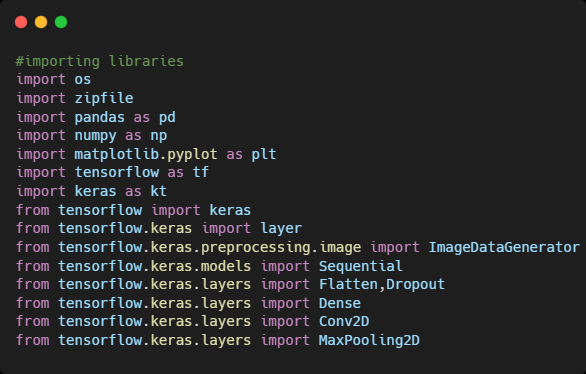
# **Problem Statement**

The main objective of this task is we are given a dataset in a zip folder of a few thousand images of cats and dogs which contains separate training and validation directories.

Using this data, we will implement a Convolutional Neural Networks (CNN) to build a Binary Image Classifier by applying Image Augmentation and Hyperparameter tuning techniques using Keras and TensorFlow to achieve the best accuracy and performance.

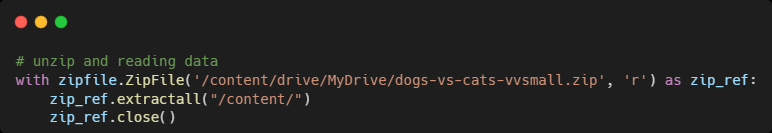
# **Importing Libraries**

* **NumPy:** For working with arrays, linear algebra.
* **Pandas:** For reading/writing data.
* **Matplotlib:** For display images and plotting model results.
* **Zip file:** For Unzipping and reading the dataset.
* **TensorFlow Keras:** We use Keras, a high-level API to build and train deep learning models in [TensorFlow](https://www.tensorflow.org/). For more information, refer to the official [Keras API documentation](https://keras.io/).

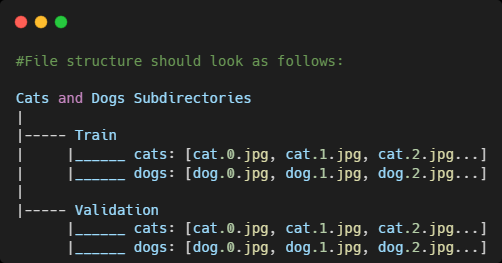


# **Unzipping and Loading the Dataset**

We are given a dataset in a zip folder of a few thousand images of cats and dogs which contains separate training and validation subdirectories, which in turn each contains cats and dog subdirectories. By importing the ZipFile library, we will unzip and read the dataset directories.



After unzipping the dataset, the file structure of cats and dogs subdirectories should look as below.



Therefore, we will divide the dataset into training and validation data frames to perform data exploration in the next step.



# **Data Exploration**

Summary of the number of images in the cat and dog training and validation dataset.

|  |  |  |
| --- | --- | --- |
|  | **Training Data Images** | **Validation Data Images** |
| **Cat** | **3000** | **900** |
| **Dog** | **3000** | **900** |
| **Total** | **6000** | **1800** |

# **Data Augmentation**

Data Augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

In simple terms, it is a method of applying different kinds of transformation to original images resulting in multiple transformed copies of the same image resulting in more data.

By applying Data Augmentation techniques, we can also avoid **Overfitting**. Overfitting occurs when we have a small number of training samples.

One solution to this problem is to augment our dataset so that it has a sufficient number and variety of training examples. With Data augmentation we generate more data from existing training samples, by applying transformations on images.

In this task, we will apply different image data augmentation techniques like

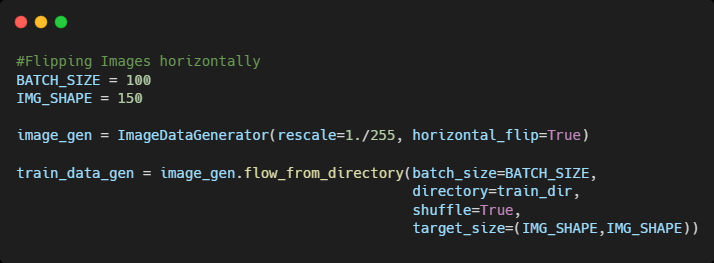
* Flipping
* Rotating
* Zooming

## **5.1 Image Augmentation with ImageDataGenerator**

The Keras deep learning neural network library provides the capability to fit models using image data augmentation using the **ImageDataGenerator** class.

### **Flipping Images Horizontally**

Let’s apply horizontal flip augmentation to our dataset and check how individual images will look after the transformation. We can do this by making “**horizontal\_flip=True**” as an argument to the **ImageDataGenerator** class.

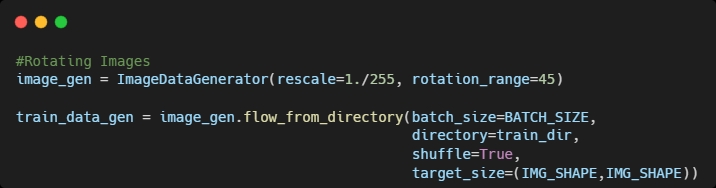


**Displaying Images after Horizontal Flipping Augmentation**



### **Rotating Images**

Let’s apply rotating image augmentation which will randomly rotate the image up to a specified number of degrees. Here, we’ll set it to 45.

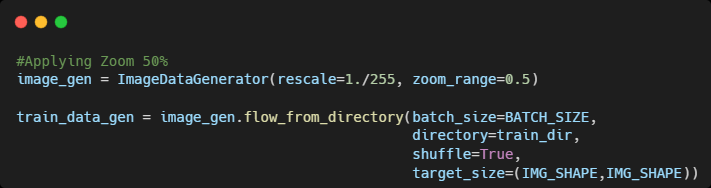


**Displaying Images after Rotating Image Augmentation**



### **Zooming Images**

Let’s apply zooming image augmentation which will zoom the image and set it to 50%.



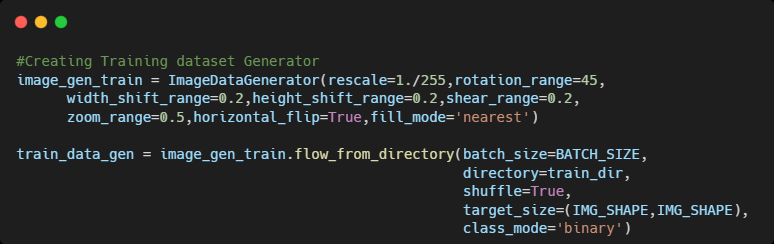
**Displaying Images after Zooming Augmentation**



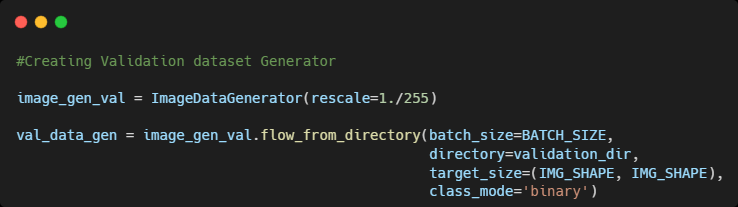
# **Model Building**

## **6.1 Creating Training Data Generator**

We will apply all data augmentation techniques together on our training dataset generator for model training and we will only add the rescaling transformation to the validation dataset generator.



## **Creating Validation Data Generator**



## **Defining the CNN Model**

* The model consists of **four Convolution (Conv2D)** layers with a **Max pool (MaxPool2D)** layer in each of them.
* **Dropout** is a regularization method, where a proportion of nodes in the layer are randomly ignored by setting their weights to zero for each training sample. This technique also **improves generalization** and **reduces overfitting**. Hence, to avoid that we will be using a Dropout layer with a probability of 50%.
* The **Flatten layer** is used to convert the final feature maps into a single 1D vector. This flattening step is needed to make use of fully connected layers after some convolutional/max\_pool layers.
* Further, we have a fully connected **Dense layer** with 512 units and '**Relu**' is the rectifier activation function that is used to add nonlinearity to the network.
* In the end, our model will be giving probabilities for two classes (Dog or Cat) we will use the **‘SoftMax’** function.



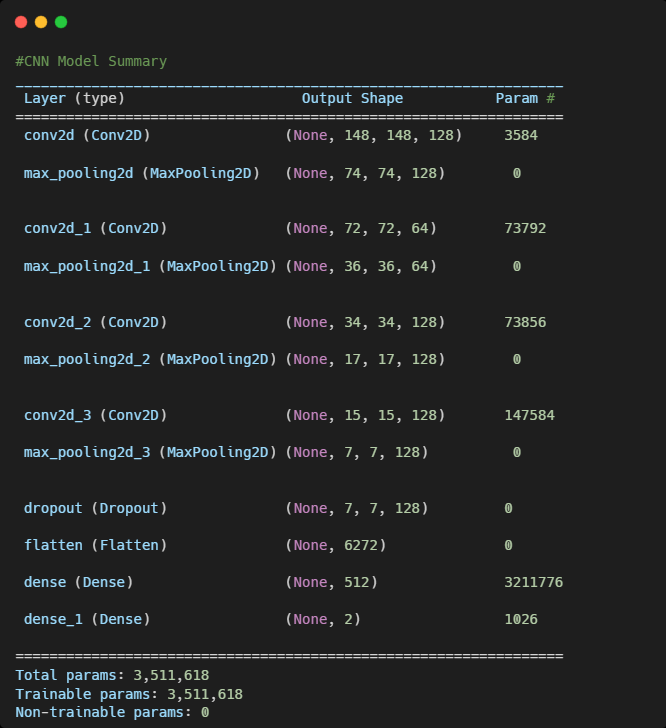
## **Compiling the Model**

Once our layers are added to the model, we need to set up a score function, a loss function, and an optimization algorithm.

* We define the **Loss function** to measure how poorly our model performs on images with known labels and we will use “**sparse\_categorical\_crossentropy**".
* The most important function is the **Optimizer**. This function will iteratively improve parameters to minimize the loss. We will go with the "**Adam**" optimizer.
* The metric function "**Accuracy**" is used to evaluate our model's performance by checking the training and validation accuracy of the model.

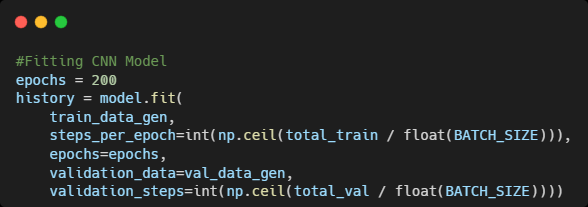
## 

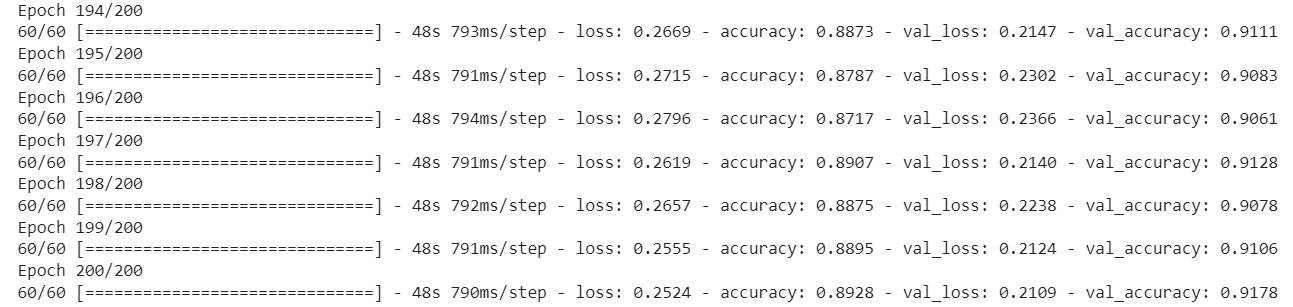
## **6.5 CNN Model Summary**



## **Model Training**

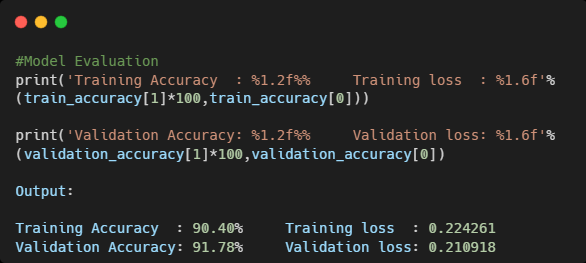
To start training, we will use the “**model.fit**” method to train the data and parameters with **epochs = 200**.





# **Model Evaluation**

As the model training is completed, the loss and accuracy metrics are displayed.



**Results:**

60/60 - 41s 681ms/step - loss: 0.2243 - accuracy: 0.9040

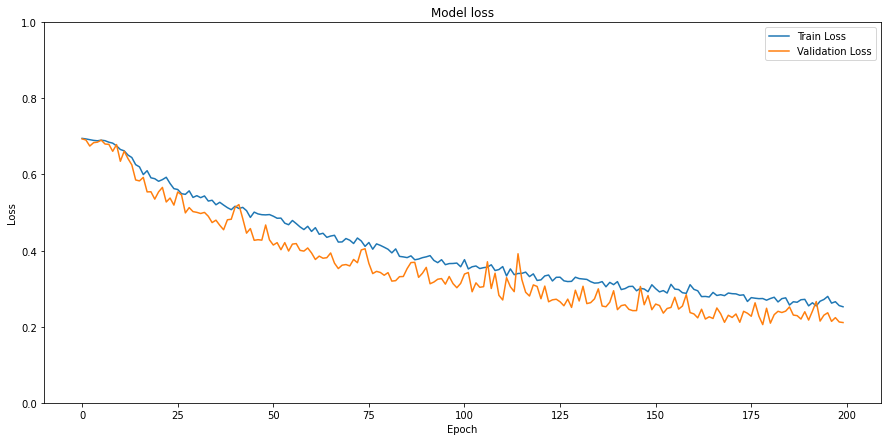
**Training Accuracy: 90.40%**

18/18 - 5s 273ms/step - loss: 0.2109 - accuracy: 0.9178

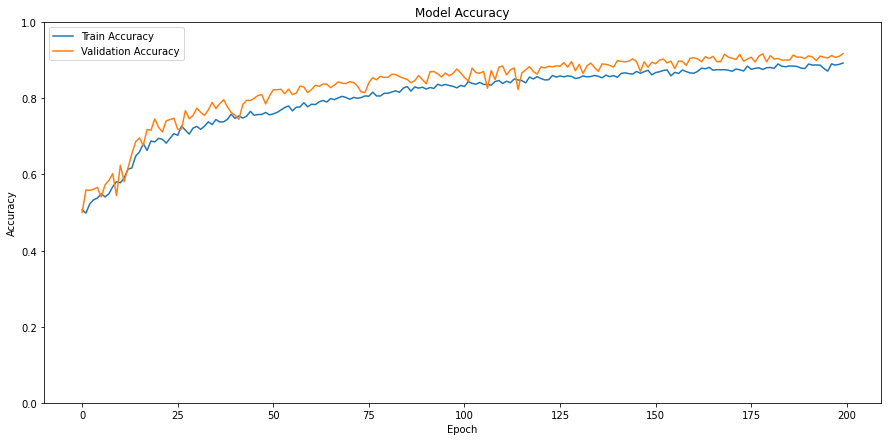
**Validation Accuracy: 91.78%**

* We can observe that the model accuracy of the **Training data** is 90.40% and the **validation data** is **91.78% (≈92%)** after 200 epochs.
* The validation accuracy is slightly greater than the training accuracy in almost every training. That means that our model doesn't overfit the training set.
* Hence, we can say that our Convolutional Neural Network (CNN) model is more **generalized** and prevented overfitting.

## **7.1 CNN Model Loss on Train and Validation Data**



## **7.2 CNN Model Accuracy on Train and Validation Data**



# **Hyperparameter Tuning of the CNN Model using Keras Tuner**

**Hyperparameter Tuning**

The process of selecting the right set of hyperparameters for the ML/DL model is called Hyperparameter tuning.

In this task, we will use the Bayesian Hyperparameter tuning technique using Keras Tuner API. In Bayesian optimization, the performance function is modelled as a sample from a Gaussian process over the hyperparameter value.

Bayesian optimization takes advantage of previously tested combinations to sample the next one more efficiently.

## **8.1 Implementation of Bayesian optimization Hyperparameter Tuning of CNN Model using Keras Tuner**

1. Hyperparameter Tuning "Learning Rate"
2. Hyperparameter Tuning "Dropout Rate"
3. Hyperparameter Tuning "Convolutional (Conv2D) Layers"
4. Hyperparameter Tuning "Convolutional Kernel (Conv\_Kernel)"
5. Hyperparameter Tuning "Dense Layers"

Python Implementation of Hyperparameter tuning of the CNN Model using Bayesian Optimisation is available on [**GitHub**](https://github.com/ChaithanyaVamshi/Image-Classification-CNN)

The best hyperparameters obtained on the dataset after rigorous hours of training and computing resources are shown below using Bayesian Optimisation hyperparameter tuning.

|  |  |  |
| --- | --- | --- |
| **Hyperparameter** | **Hyperparameter Values** | **Best Hyperparameter Values** |
| **Learning Rate** | [0.0001, 0.001, 0.01] | 0.0001 |
| **Dropout Rate 1** | Range (0.0,0.5,0.05) | 0.0 |
| **Dropout Rate 2** | Range (0.0,0.5,0.05) | 0.0 |
| **Conv2D Layer 1** | Range (16,128,32) | 112 |
| **Conv2D Layer 2** | Range (32,128,32) | 96 |
| **Conv\_Kernel 1** | Range (3,5) | 5 |
| **Conv\_Kernel 2** | Range (3,5) | 5 |
| **Dense Layer** | Range (16,128,32) | 112 |

## **Model Training Neural Network with the Best Hyperparameters**

Building the model with the optimal hyperparameters obtained and training it on the data for **epochs = 125**.

# 

## **Model Evaluation Neural Network with the Best Hyperparameters**

As the model training is completed, the loss and accuracy metrics are displayed.

**Results:**

60/60 - 48s 790ms/step - loss: 0.3876 - accuracy: 0.8238

**Training Accuracy: 82.38%**

18/18 - 6s 322ms/step - loss: 0.3884 - accuracy: 0.8272

**Validation Accuracy: 82.72%**

## **Overall Model Accuracy Results Summary**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CNN Model** | **Epochs** | **Hyperparameter Tuning** | **Training Accuracy** | **Testing Accuracy** |
| **Model 1** | **200** | **No** | **90.40%** | **91.78%** |
| **Model 2** | **125** | **Yes** | **82.38%** | **82.72%** |

**Overall Model Accuracy Results Summary**

* The first CNN model is trained for **200 epochs** and the Accuracy given by the Training set is **90.40%** and the Accuracy given by the Testing set is **91.78%. (≈92%)**
* The model with the best hyperparameters is only trained for **125 epochs** and the Accuracy given by the Training set is **82.38%** and the Accuracy given by the Testing set is **82.72%**.

Deep Learning algorithms can handle large amounts of data but it requires a large amount of training and computing resources.

**Since the training process is more time consuming and I believe if we can increase the number of epochs from 125 to 200 to the hyperparameter model we can see an improvement in the accuracy.**

**Hence, we can summarize that both the neural network models are more generalized (learned well) and prevented overfitting.**

# **Conclusion**

In this blog, we discussed how to approach the image classification problem by implementing the Convolutional Neural networks (CNN) model using Keras, TensorFlow with **92% accuracy**.

We can explore this work further by trying to improve the accuracy by using advanced Deep Learning algorithms and hyperparameter tuning techniques.

# **References**

[1] <https://www.tensorflow.org/tutorials/keras/classification>

[2] <https://keras.io/guides/keras_tuner/getting_started/>

[3] <https://keras.io/api/keras_tuner/tuners/bayesian/#bayesianoptimization-class>

[4] <https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/>