# NEURAL NETWORKS: BINARY CLASSIFICATION

CATS & DOGS DATASET

ID: 991043
LAST NAME: RASHAD
Name: Amr Mohamed Nazih Mohamed

# Abstract

The aim of this project is to develop a neural network to perform binary classification on the given Cats & Dogs images dataset. The main type of neural network that will be focused on is convolutional neural network (CNN), where I try exploring different network architectures, tunning various hyperparameters and applying 5-fold cross validation in order to compute the risk estimates of the selected model. To determine the best performing CNN, I have applied different models and introduced regularization methods in order to avoid overfitting.
I come to the conclusion that the best experimented performing CNN architecture is composed of 4 convolutional layers with image size 100x100.

*I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.*

# 1. Introduction:

Neural networks is a reflection of how the human brain behaves, where the aim is to recognize patterns and solve problems in various fields such as machine learning. This project is an experimentation with the aim of untangling a classification problem, specifically binary classification. A very common and wildly popular dataset has been used in this project, which is cats and dogs dataset. This dataset is composed of 25000 images, half of which are cats images, and the other half are dog images.

There are several types of neural networks, however this project focuses only on convolutional neural networks (CNN). A specific CNN architecture has been explored, with different image sizes and different regularization methods.
This project has been composed as a pathway, where at each step we determine where we should go next.

We can break this pathway into 5 key points:

- Cleaning the dataset from any broken/corrupted images. The final count of cats and dogs images is 12383 cats images and 12349 dogs images.
- Splitting the dataset into train and test sets, 80/20 respectively.
- Performing trial models, where overfitting has been encountered and resolved by introducing 2 regularization methods, data augmentation and dropout layers.
- Experimenting with 2 different image sizes, 64x64 (possible minimum size) & 100x100
- Experimenting with 3 & 4 convolutional layers in order to determine which approach results in a better model performance.
- Tunning various hyperparameters to achieve highest accuracy.
- Applying 5-fold cross validation on the best performing CNN model.

The CNN models are compared by observing the accuracy score of each model. We also investigate how each model is performing on the train set and the validation set by comparing the accuracy scores of both sets in each epoch, this helps us determine if the model is underfitting/overfitting.

# 2. Theoretical Definitions & Concepts:

## i. Binary Classification:

Binary classification is the process of categorizing an observation into one of two classes, i.e., 0 or 1. For instance, in this project, our two classes are cats and dogs. Therefore, the aim would be to classify a given image as either a cat or a dog. Since we're going to train our model, we would need to apply a loss function as a measure of howe well the neural network models the training data, this will be performed by comparing the target and predicted output values. Because the project discusses a binary classification issue, we are going to apply Binary Cross-Entropy as our loss function.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

*Figure 1*

Figure 1 is a representation of the Binary Cross-Entropy formula, which is a computation of the mean of the negative log of the predicted probabilities, where $y_i$ is the label representing each image (cat or dog), $p(y_i)$ represents the predicted probability.

## ii. Neural Network:

A neural network is a computational learning system where input data are translated into a desired output through the use of network of functions. It's one of many tools of machine learning algorithms and can be applied in many real-life problems such as image classification, which is the task our project is addressing.

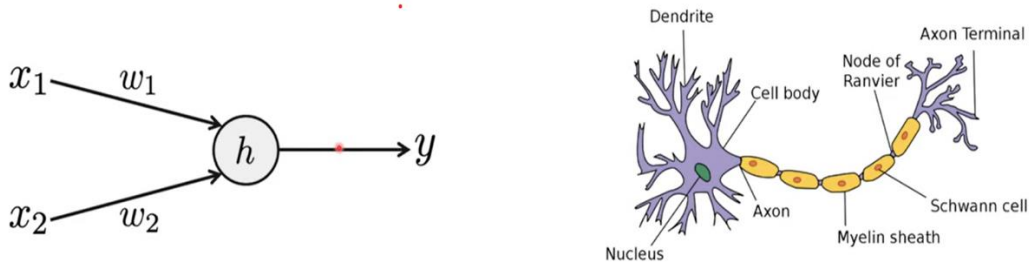


*Figure 2*

The artificial neural network is an inspiration from the biological one. In figure 2, we are able to see that each neuron is an attempt to represent the biological neuron, meaning the artificial neuron has the same architecture as the biological one. Where first the biological neuron starts by accumulating inputs from all its surroundings, then they are weighted. Then there is a threshold, where it would channel the weighted inputs who exceed the threshold to a neighboring neuron that are connected to this neuron. This is actually where the similarity stops. After that, the biological neuron becomes much more complex than the inspired artificial neuron.

In figure 2, the neuron is represented as '$h$', $x_1$ & $x_2$ are inputs, $w_1$ & $w_2$ are their weights and y is the output. Where $h = W^T X$, w is a matrix of weights and x is a matrix of inputs.

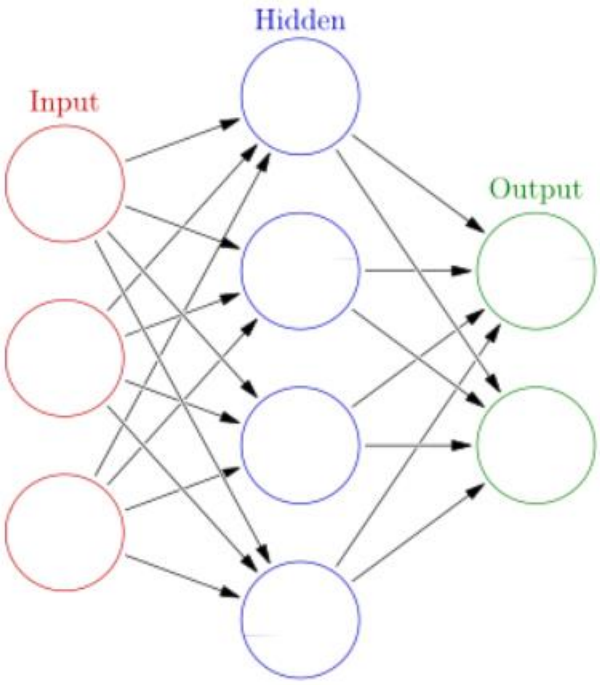


*Figure 3*

Figure 3 is a representation of a neural network, each input is connected to all the neurons, and all the neurons process the same inputs in parallel just with different weights. These neurons are actually grouped into a single group called a layer.
This can be described on a higher level as one function that takes vector x as input and predicts/produces a new vector y, as represented in figure 4.

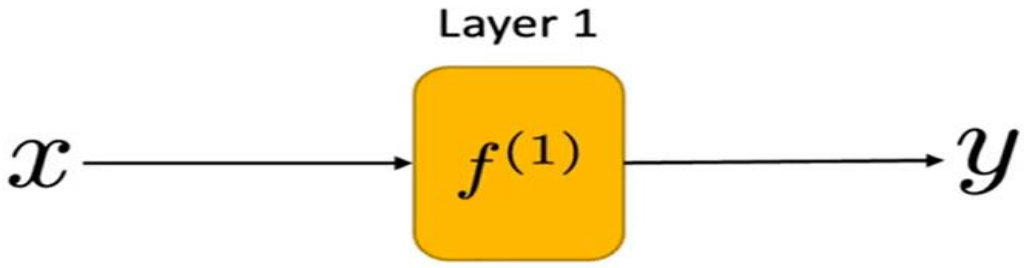


*Figure 4*

If we were to add a second layer, then we would pass the output of the first layer into the second layer and have all the connection that we would have for a normal input and then the second layer predicts the output, as seen in figure 5.

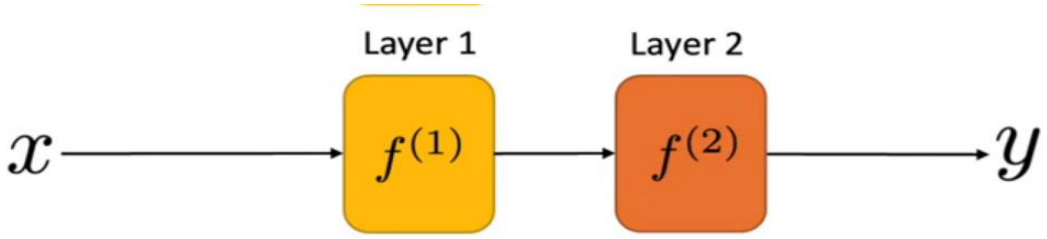


*Figure 5*

We can scale up the first layer to the point where we don't need to add additional layers, however this would be computationally expensive as the number of weights is equal to the number of inputs times the number of hidden units, also it's hard to find weights that work in practice. Additionally, what's interesting to note is that if we're trying to imitate the human brain, then we cannot follow/apply this idea since the human brain doesn't process everything in parallel, it's actually built in more of a hierarchical way. Hence, we need to add more hidden layers. This process is called deep neural network.

## iii. Image Transformation:

We start by transforming our images into grayscale and in order to use it as input to our model we would need to transform it into numbers. We assign a number to each pixel and the values of colors are rescaled between 0 and 1, we divide the actual color value by 255, where 0 means black and 1 means white and everything else is between these two numbers. Once we assign these numbers to our pixels, we would then have a matrix, hence a mathematical representation of the image that can be used as an input to our model, as shown in figure 6.

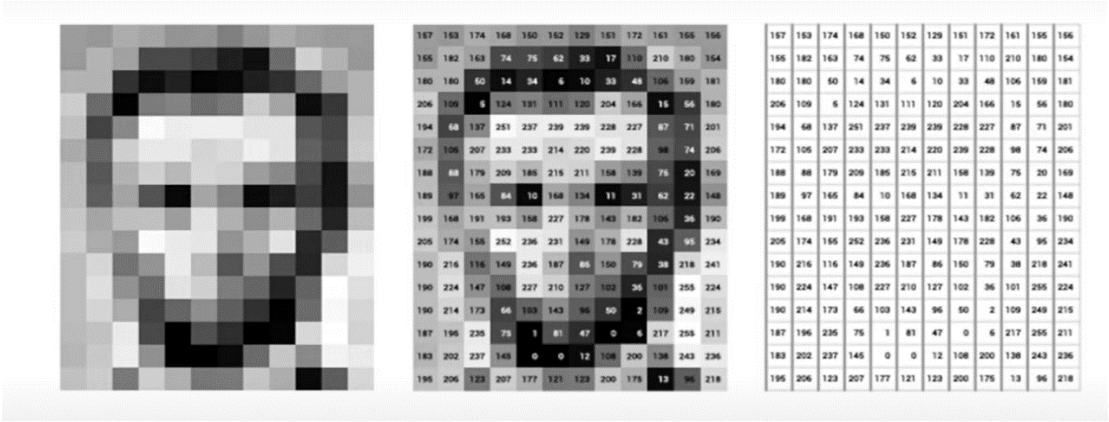

*Figure 6*

## iv. Convolutional Neural Network & Convolutional Layer:

We discussed the concept of a standard neural network, where each pixel is considered as input to our model, as shown in figure 7. We can apply such model; however, it results in an extremely high number of parameters, where in some cases we won't be able to have either enough training samples to train the model nor the memory to store such model. Hence, the aim would be to reduce the number parameters while still having a good model.

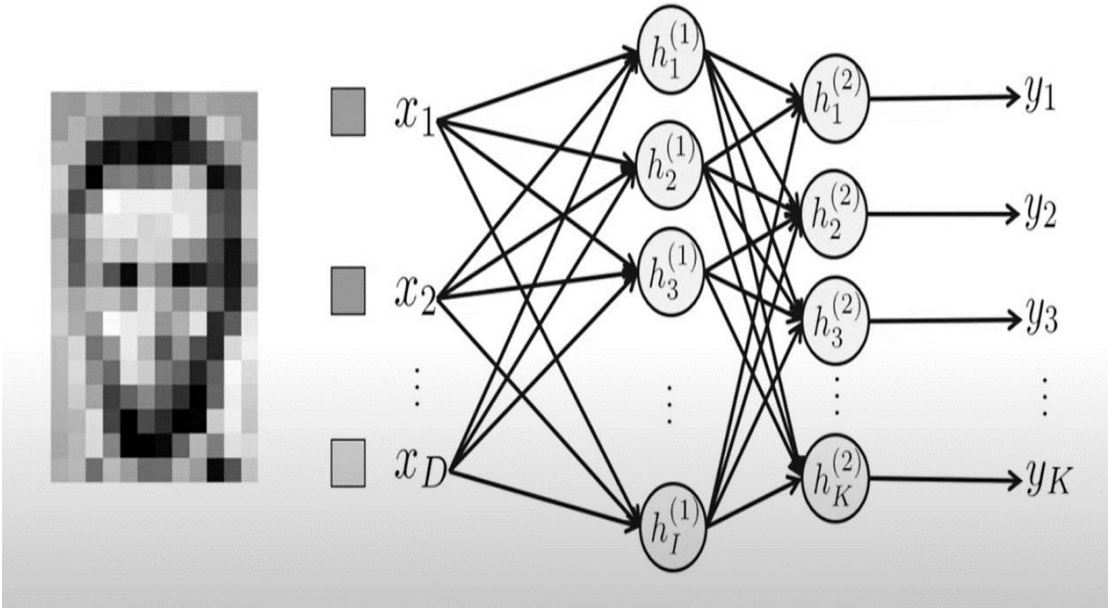


*Figure 7*

Therefore, the idea would be instead of having weights from all pixels to a single neuron, we can connect part of the information be connected to the neuron, meaning we have a single neuron that only gets weight connections from this local area, in other words it's sensitive to local feature at this specific position. We then slide the local area one pixel at a time and that would give us another unit that is sensitive to the same feature but at a slightly different position. We keep on sliding one pixel at a time until we cover the whole image, as shown in figure 8.

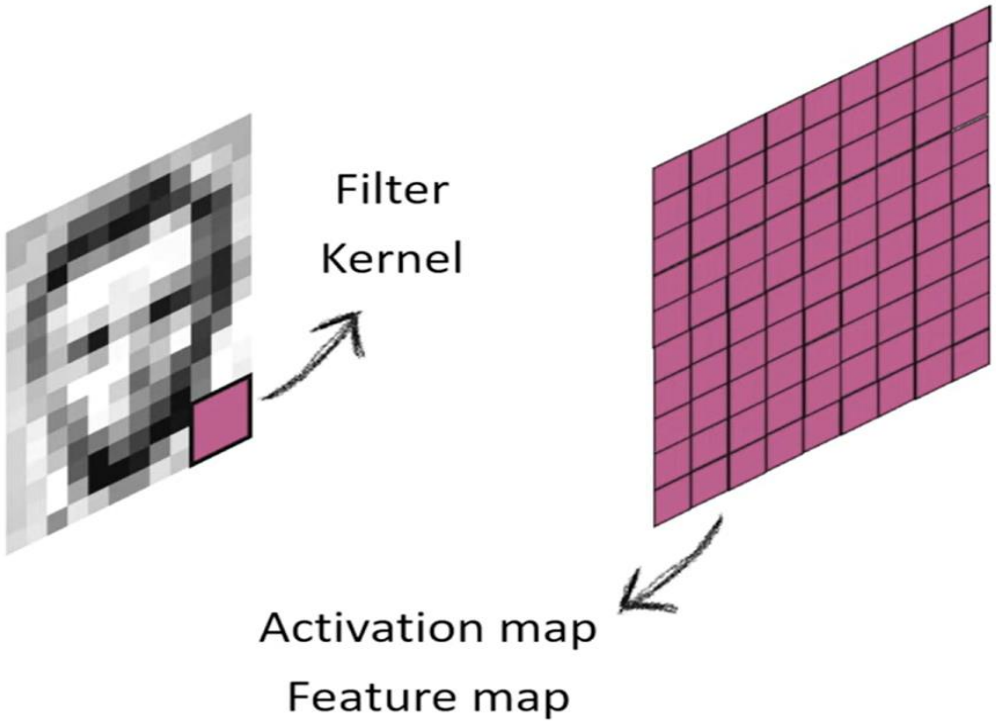


*Figure 8*

This operation is called the convolution operation, basically sliding a set of weights over the input matrix to generate a new matrix. The set of weights connections are commonly referred to as filter or kernel. The output of the convolution operation is often called activation map or feature map.

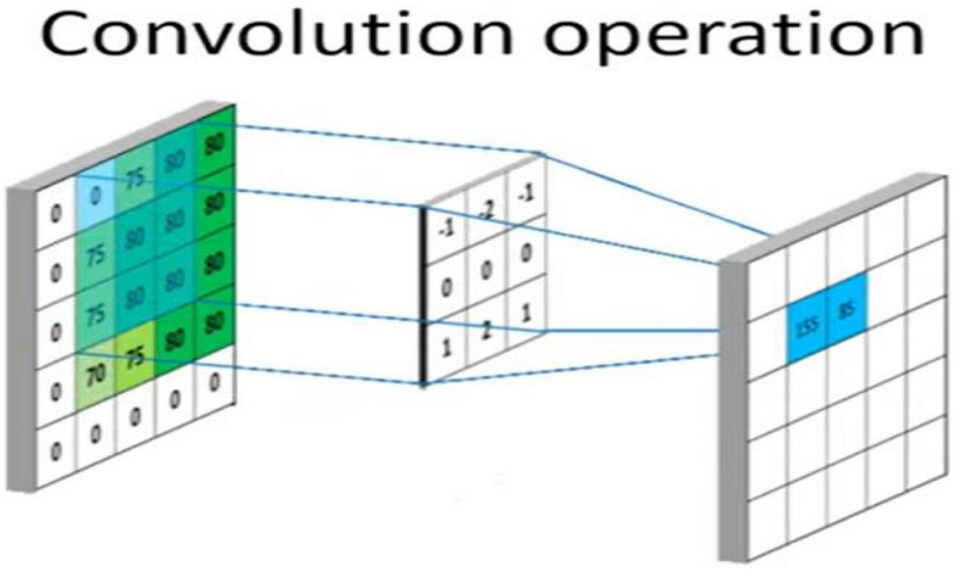


*Figure 9*

The main motivation of using these weight connections was because our original neural network had too many parameters. If we apply a 3x3 filter, like we did in our model, we will have 9 parameters per position. So, looking at local information pays off in terms of number of parameters but also in terms of performing tasks. So, we end up with a model with less parameters but also performs very well.
The convolution layer is composed by the following parameters:

- Filter which is a single neuron that sweeps through the image and produces an activation map, as shown in figure 10.

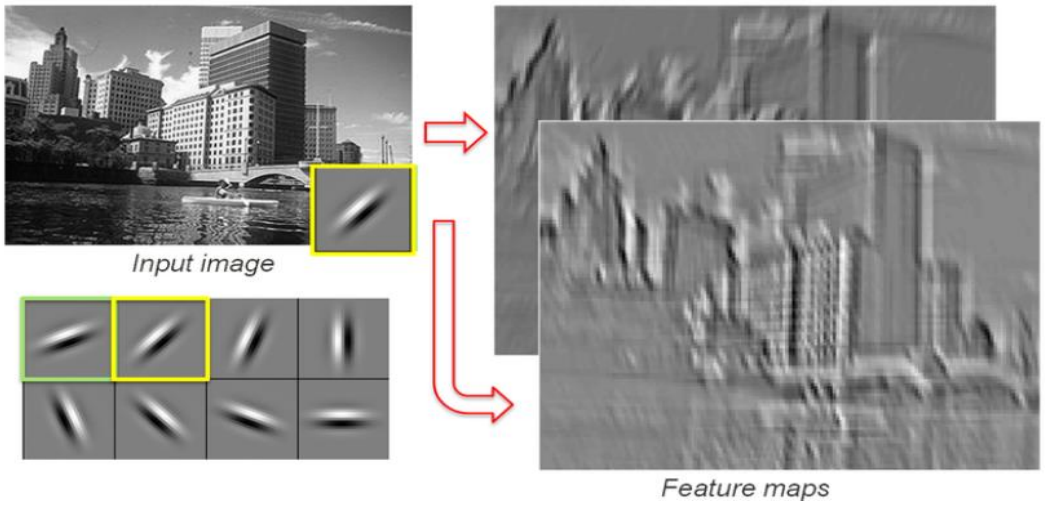


*Figure 10*

- Filter dimensions, in our case it's 3x3.
- Number of filters, this parameter determines the depth of the activation map, depth can be also described as number of channels, as shown in figure 11.

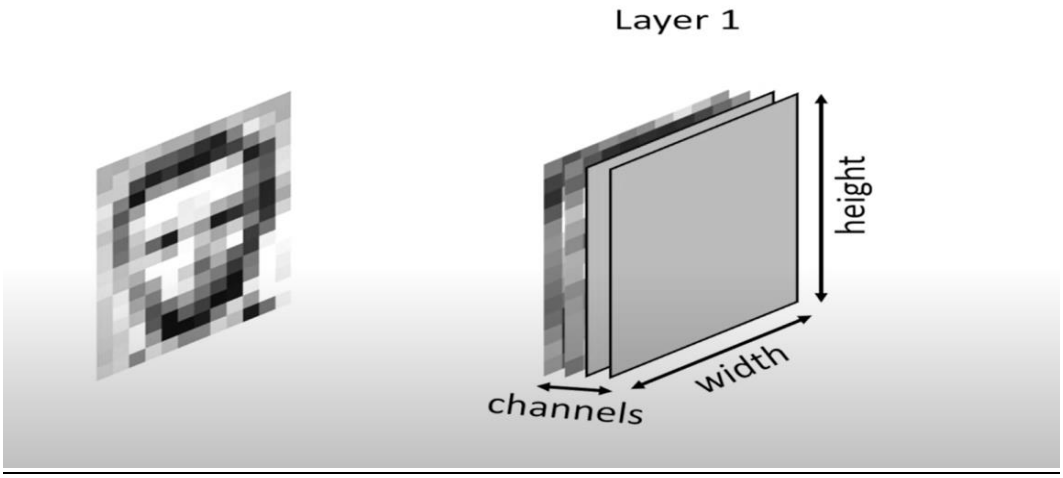


*Figure 11*

- Stride, which is a parameter that describes the movement of the filter over the given image. If stride is set to 1, then the filter would move by one pixel at a time. The output activation map is inversely proportional to the stride number, meaning the greater the stride gets the smaller the activation map will be.
- Activation function, there are several types however we will be discussing and use the ReLU function. As shown in figure 12, ReLU function is computationally efficient as it converts all negative inputs to zero and the neuron does not get activated, hence few neurons get activated.

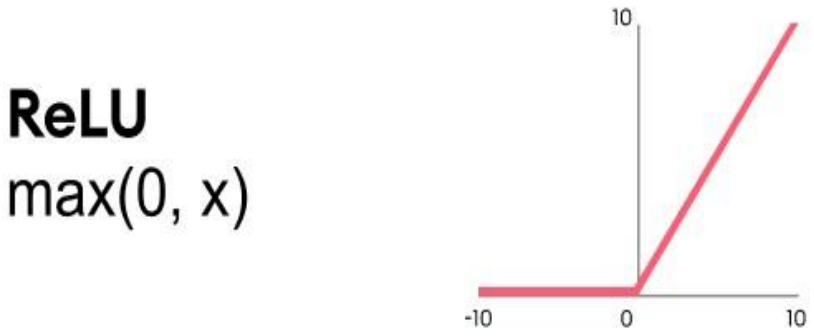


*Figure 12*

## v. Pooling Layer:

The aim of this layer is to reduce the number of parameters and the computation in the network. However, another benefit to adding this layer is to avoid overfitting by reducing the spatial size of the network. There are two types of pooling layers, average pooling, and maximum pooling layer. In this project we have applied max-pooling layer. This layer has filters as well, where at each stride, the filter slides through the given input and takes only the maximum parameter, as shown in figure 13.

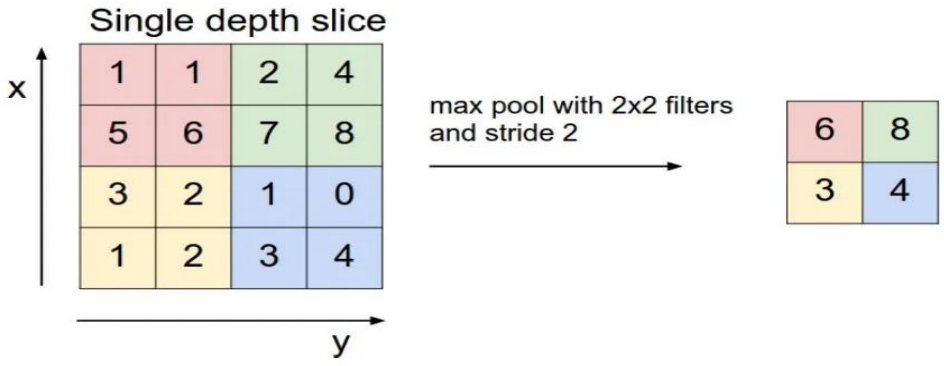


*Figure 13*

## vi. Dropout Layer:

Dropout is a regularization method, where the aim of this layer is to help the model to avoid being overfitted. The term ‘drop’ simply means dropping out nodes, as shown in figure 14. Assuming we have a dropout layer with probability 0.2, which is the case in our first dropout layer, this means that this layer will drop or discard 20 percent of the input nodes. Generally speaking, 0.5 is the most optimized keep probability, which is used in our second dropout layer.

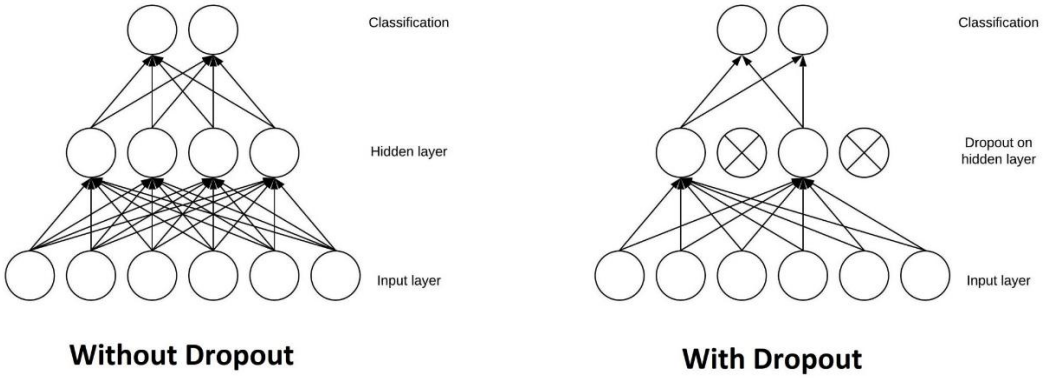


*Figure 14*

The result of the dropout method is making the layer look-like a layer with different number of nodes, as shown in figure 14, this introduces more noise in the training process, which in consequence reduces interdependent learning amongst the neurons and focuses more on generalization.

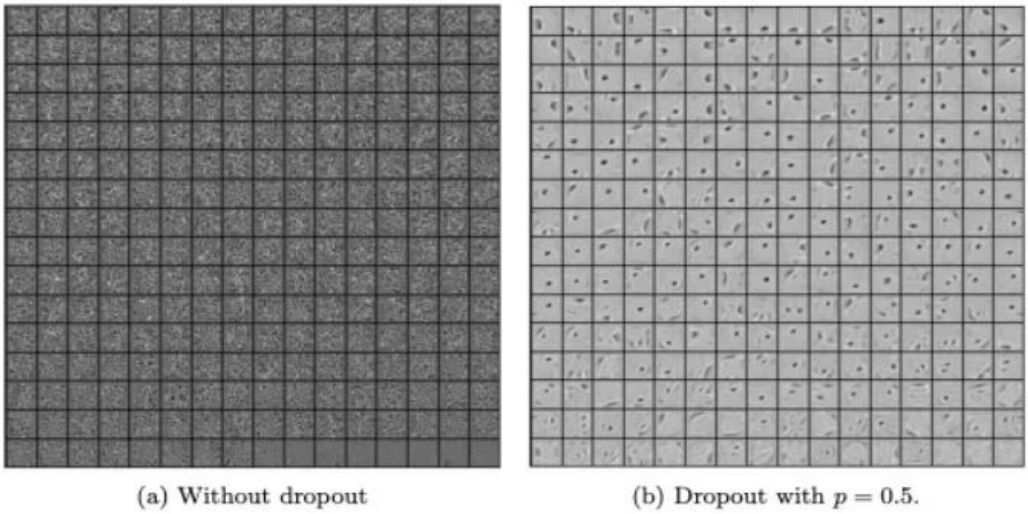


*Figure 15*

## vii. Flatten Layer:

Simply put, the flatten layer transforms the 2D arrays from pooled activation maps into a one-dimensional vector, which is then used as an input for the dense layer, as shown in figures 16 and 17.

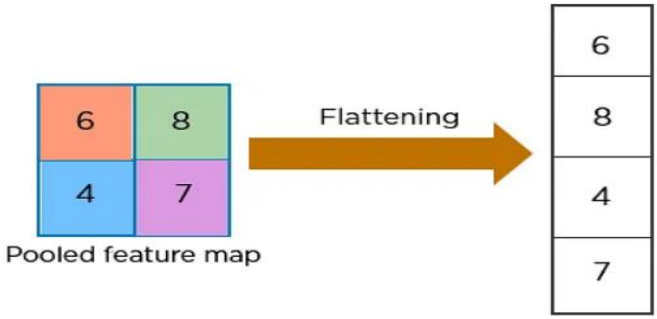


*Figure 16*

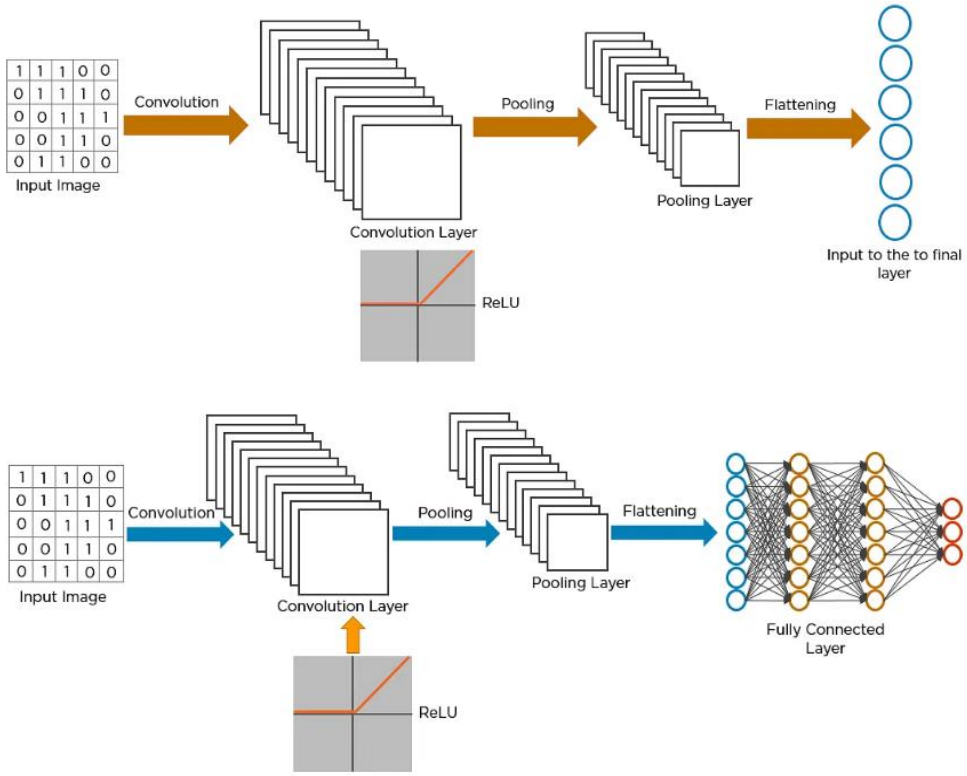


*Figure 17*

viii. Dense Layer:

Dense layer, also named fully connected layer, is where each neuron is connected to all the neurons from the previous layer, in this case the flatten layer, as shown in figure 17. This layer is used to classify images based on the given inputs from the previous layers. Similar to the convolutional layer, the dense layer has an activation function as well. In our case we have decided to apply the sigmoid function, the reason is it results in output values between 0 and 1, as shown in figure 18, which can be used in a binary classification problem, our project. This in turn can be interpreted as the probability of a given image belonging to a specific class.

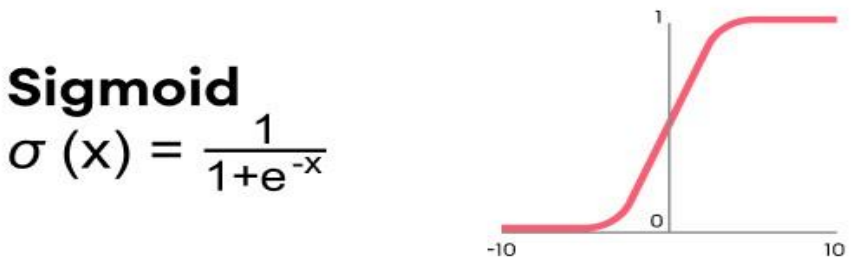


*Figure 18*

# 3. Project Layout:

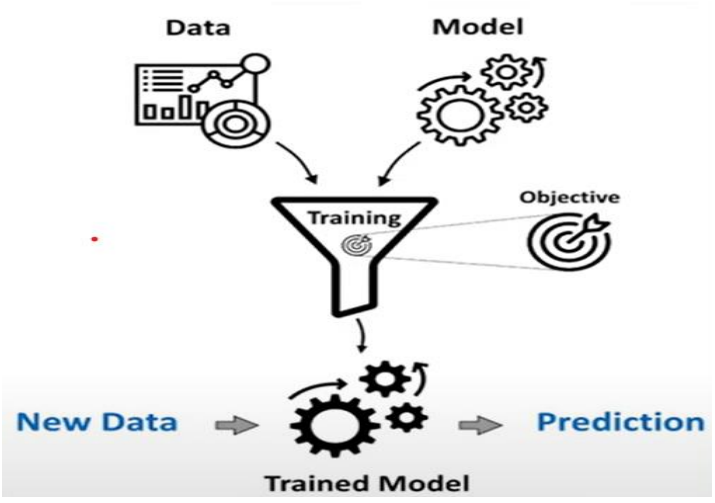


*Figure 19*

We can break up the process of this project into three main parts, as shown in figure 19. The first one is to specify the model, meaning create the model specifying the number of layers, filters, etc. The next main part is to define the objective, in our case this is a binary

classification problem, therefore we need to apply the binary cross entropy loss. The final main part is we train our model. We can visualize this process through two graphs. One represents the loss, and the other represents the accuracy of the model, as shown in figure 20. It's usually epochs (x-axis) versus the loss/accuracy scores. Epochs are iterations over the entire training.

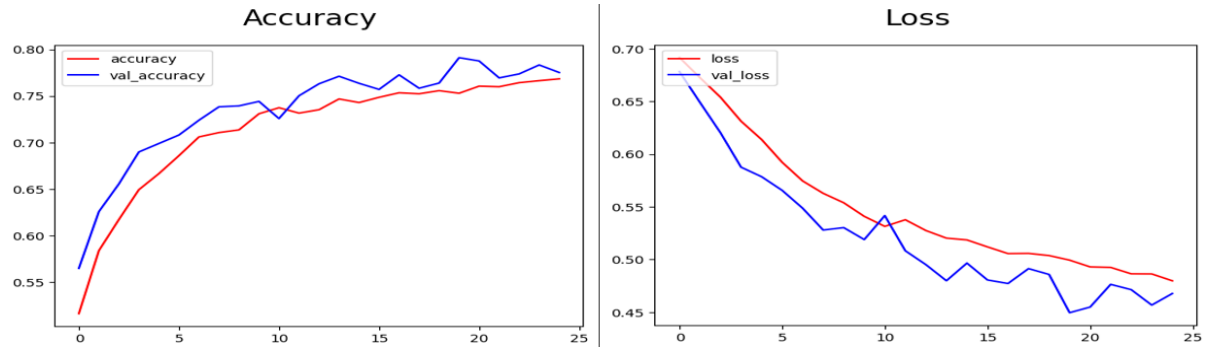


*Figure 20*

However, usually we have an additional problem when we try to train our model. This is because our goal is not only to find a model that fits the training data well but also, we want the model to predict/perform accurately on new data, and to measure that we introduce a second set, validation set, where we test on unseen data and the parameters are not actually optimized, as shown in figure 20. This is actually the most challenging part, because we usually run into one of two problems, underfitting and overfitting, meaning we are trying to find the best generalization to unseen data.

Overfitting is encountered when our model is too complex, meaning it memorizes the training data very well and is not able to fit the actual underlying function anymore. Meaning the model won't be able to predict the validation points that don't lie exactly where the training data lies. Hence, this results in poor generalization and produces a graph similar to the one presented in figure 21, where the validation set curve starts to become more of  a bell-shaped curve.

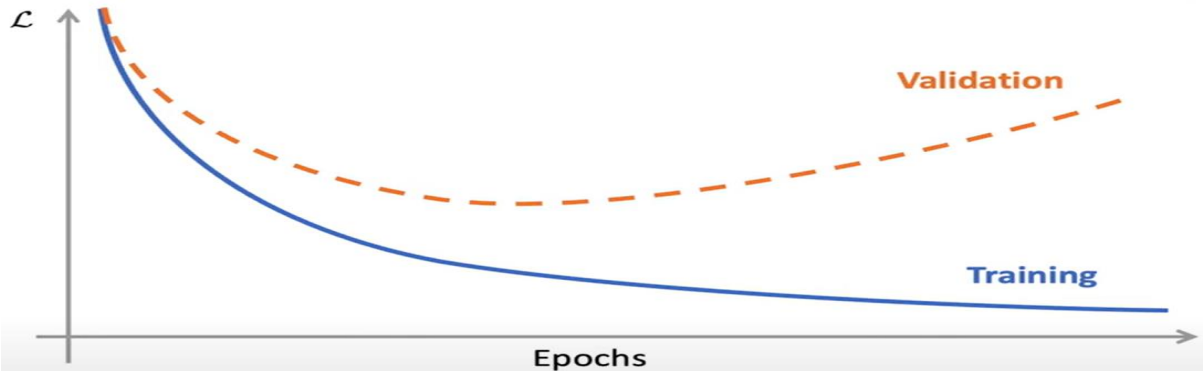


*Figure 21*

Underfitting is actually the opposite to overfitting, where we have too little capacity and we cannot really learn anything, not even the training data. Therefore, the model cannot fit the training data and has poor performance on both training and validation, as shown in figure 22.

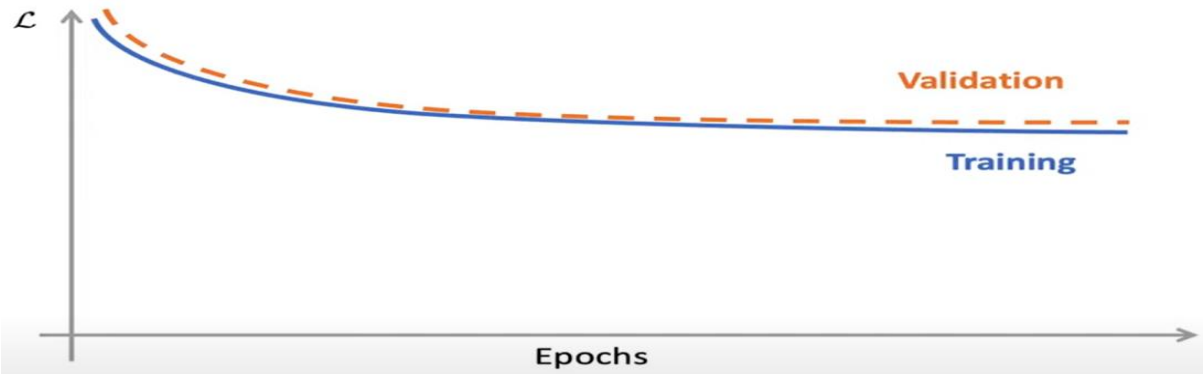


*Figure 22*

A good fit is actually where we have the right capacity, and we are able to fit the underlying function and hence perform predictions on new unseen data as well. Where we end up with both validation and train loss somehow aligned or are close to each other, as shown in figure 23.

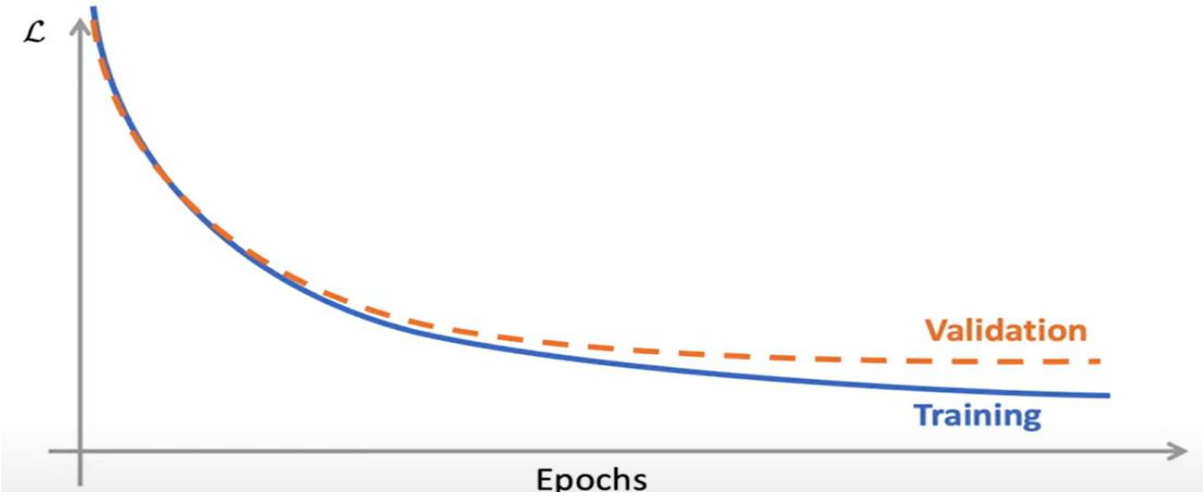


*Figure 23*

This is of course the aim when we train our model, achieving a good fit and avoid overfitting/underfitting. One way to avoid overfitting is by adding more variations into the input data, augmenting the data, as shown in figure 24, this can be done be rotating the image for example or zooming in on the picture or flip the image horizontally, etc.

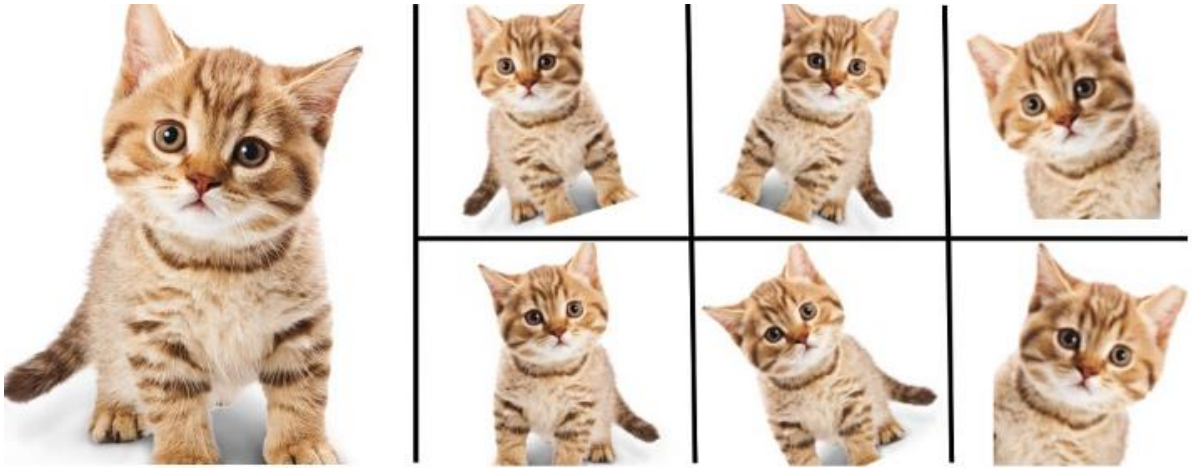

*Figure 24*

This would help to classify more variations of the same images which would lead to a decrease in overfitting. Another solution would be dropout, as discussed earlier in section 2.

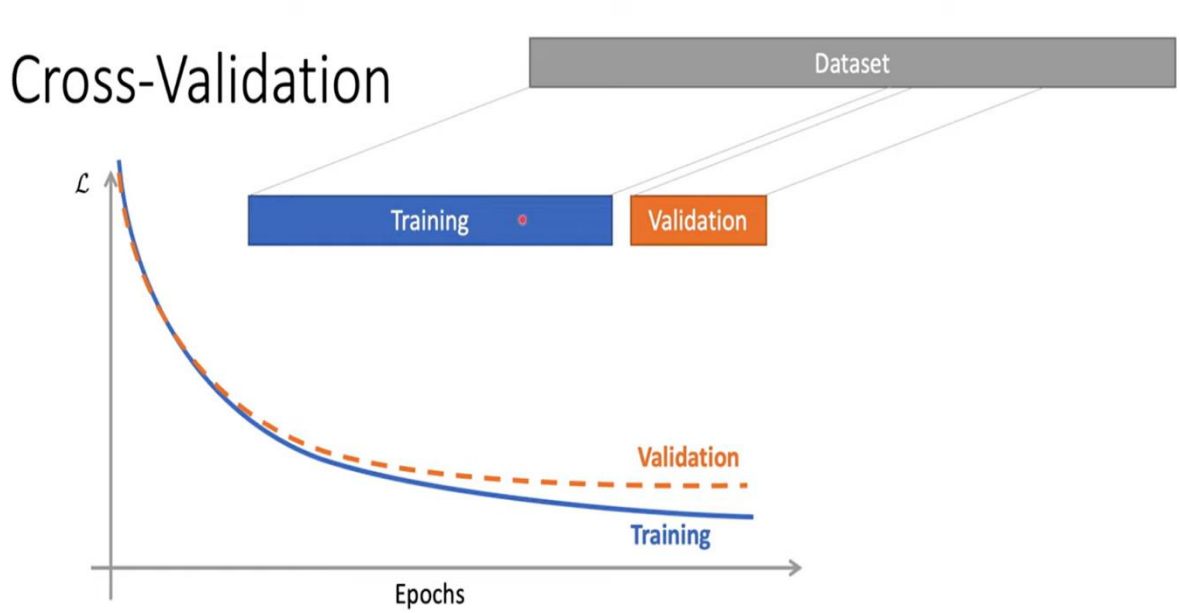


*Figure 25*

To achieve the best true generalization, we would need a test set that is separate from training and validation data to report the performance at the end, as shown in figure 25. We can then apply cross validation. Simply put, cross validation is the process of splitting the data into smaller samples and evaluating how well the model can predict the outcome of unseen data. K-fold cross validation is the same concept, however we perform the cross-validation k times, and we then compute the mean of the k performances of the model, as shown in figure 26.

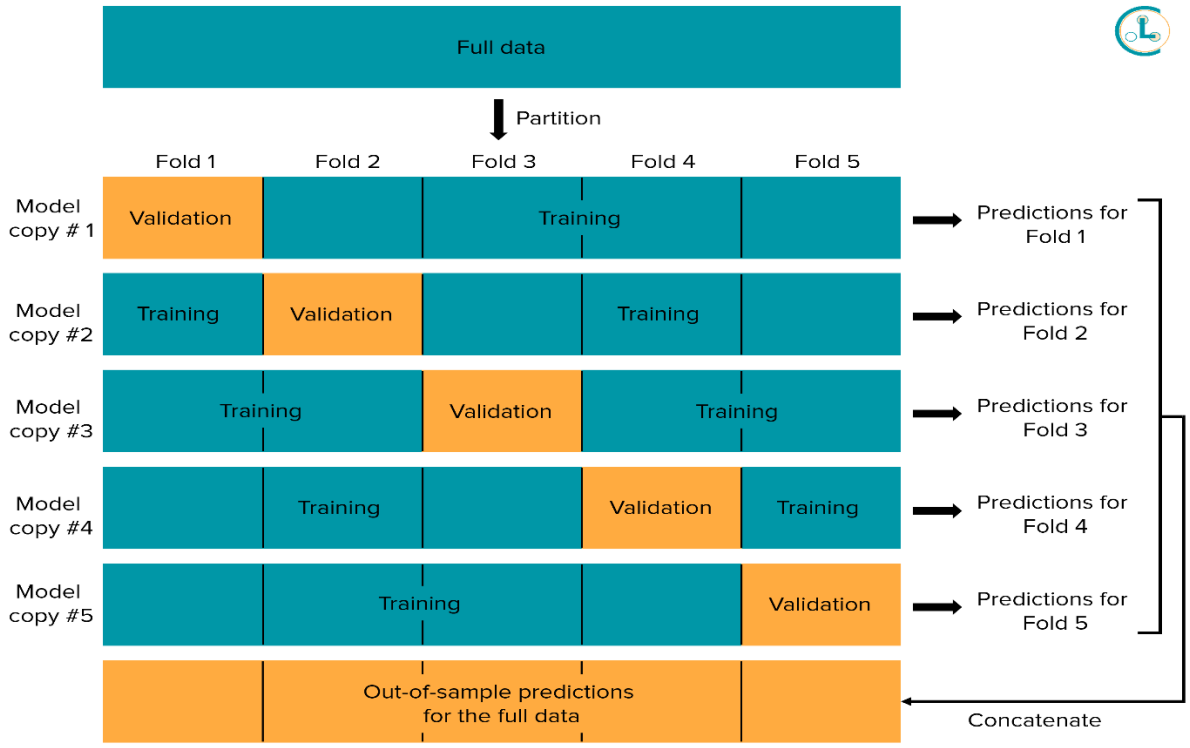


*Figure 26*

In our project we apply 5-fold cross validation, where in each fold we select the best model, and we would then evaluate the best model of each fold on the test set and finally we would average the scores together and get an accurate more generalized performance report of the model.

# 4. Models Description:

Image classification is a very well-known and solved task, where there are extensive research papers addressing such problems with the development of deep learning models. Hence, one would be interested to learn about the different types of model architectures that has been applied and might get an inspiration from. For instance, ImageNet challenge competition is a good representation of the different deep learning models applied on the given ImageNet dataset. This is an annual competition that first took place in 2010, where the aim is to perform image classification and the top 5 models with the lowest error rate are announced as winners.

As shown in figure 27, we notice that human performance has an error rate of 5 percent. We can observe that the deep learning models achieve lower error rates with each year, for instance, AlexNet was the first convolutional neural network applied successfully to this task with error around 16 percent, while in 2015, ResNet was able to achieve an error rate that was better than the original human performance.

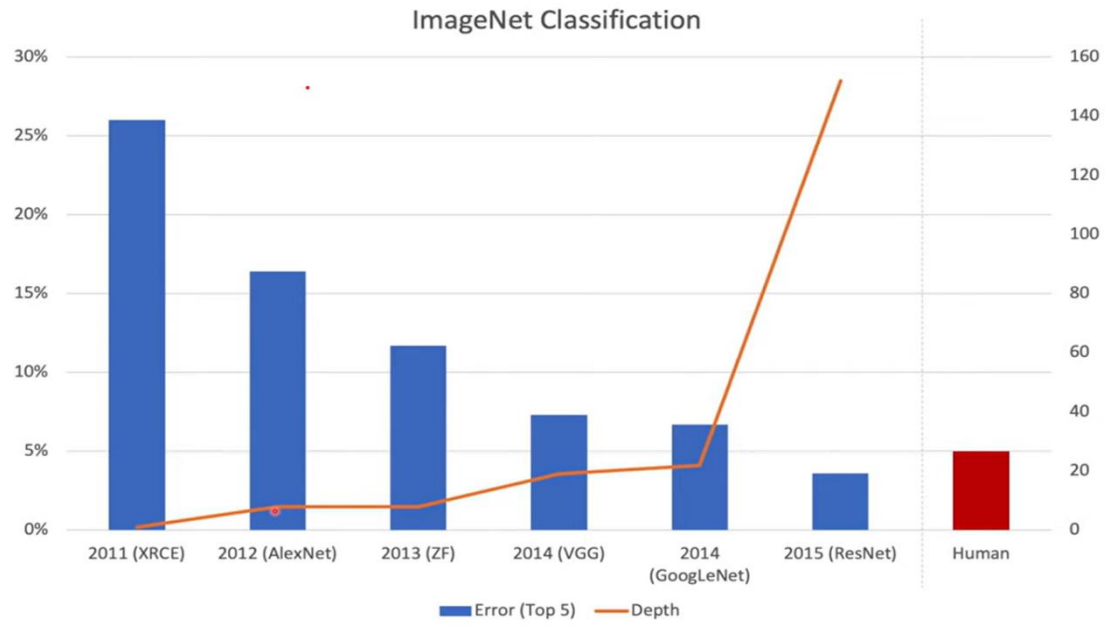


*Figure 27*

We can also observe from figure 27, that the number of hidden layers starts to grow beginning from AlexNet and it increases massively in 2015, ResNet model. Figure 28 represents the up-to-date ImageNet classification deep learning models performance accuracy.

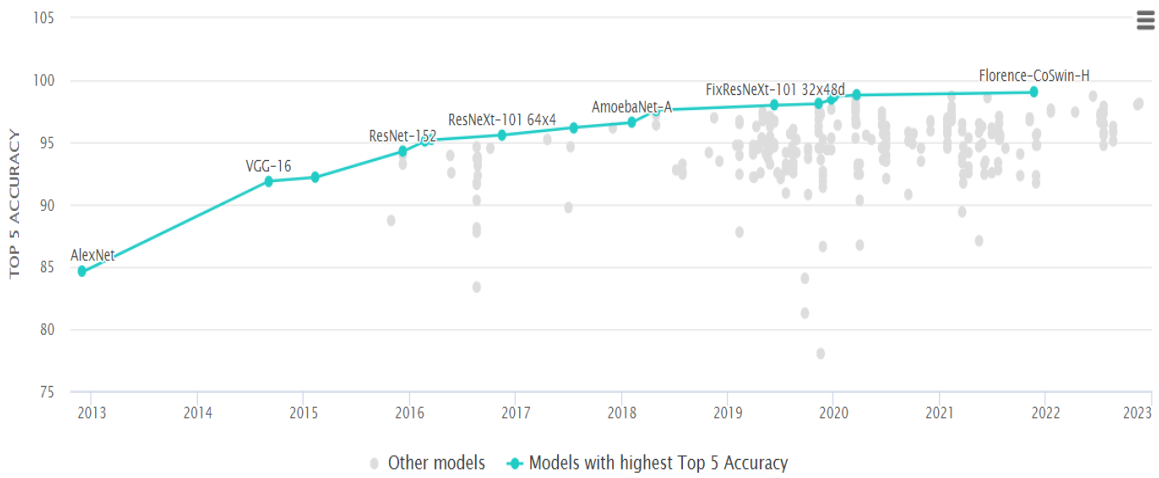


*Figure 28*

The architecture of our models was inspired from the VGG deep learning model, having 3x3 as the size of our filters and our convolutional layers use ReLU activation functions, with a max-pooling layer after each convolutional layer. The number of convolutional layers was one of the factors that we have altered while developing the models. The reason behind following the VGG architecture is because it is considered to be one of the most popular image classification models. Figure 29 is a representation of the VGG architecture, our models are inspired by this widely used architecture and they are extremely similar to it.

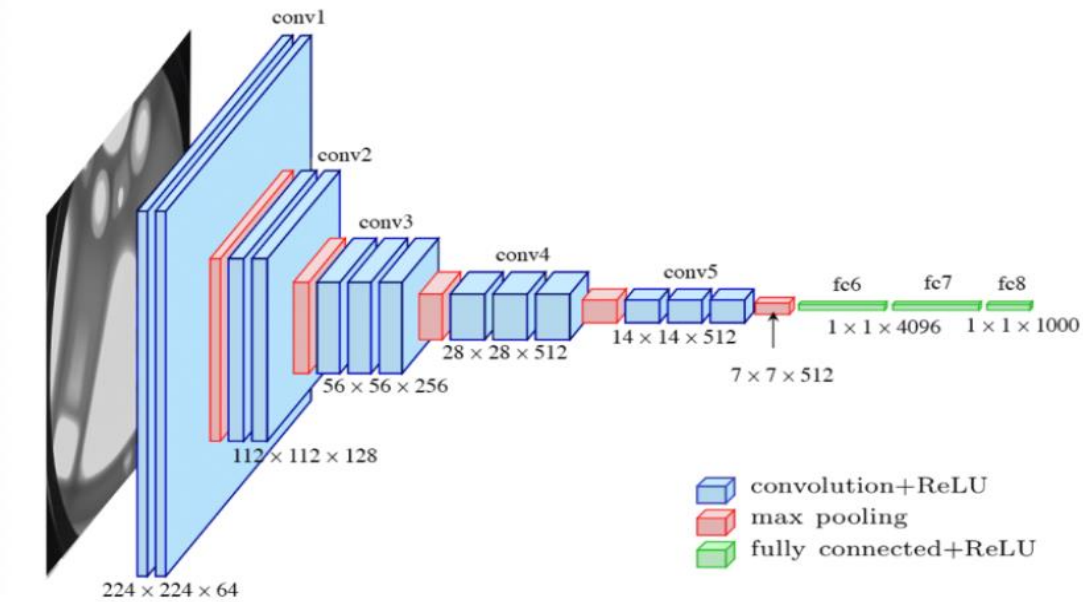


*Figure 29*

The typical number of convolutional layers in a CNN is 3, hence our models will have three convolutional layers and we will evaluate whether the model performs better when adding an additional convolutional layer. As we have discussed earlier a large image size can lead to a large amount of parameters, and because we are operating with a simple computer, i.e. not using a GPU, we will try to minimize the image size as much as possible while still aiming to achieve high accuracy. Hence, the initial image size is 100x100. From here we start developing our models and with each one we try to improve the accuracy and deal with the encountered problems. It's also worth noting that we have split the dataset into train and test sets, 80/20 respectively.

## i. First Model - Initial Model:

Our initial model is very simple, where we only have three convolutional layers with filter size 3x3, number of filters 32 ,64, 128 respectively and stride 2. After each convolutional layer we apply a max-pooling layer with filter size 2x2. Finally, we have the flatten layer accompanied by the dense layer with 128 filters and output layer.


*Figure 30*

The number of iterations over the entire training set, i.e., epochs, is 25, we can clearly notice that our model is heavily overfitted, as shown in figure 30. Hence, we will need to apply some regularization methods.

## ii. Second Model – Adding Dropout Layers:

The first regularization method that will be implemented is the dropout method, where we will be adding two dropout layers, one after the last convolutional layer with probability 0.2 and one after the dense layer with probability 0.5. Our second model has exactly the same architecture with the addition of the dropout layers.


*Figure 31*

As shown in figure 31, we can still observe that our model is heavily overfitted, therefore an additional regularization method is required.

## iii. Third Model – Implementing Data Augmentation:

The second regularization method we will be applying is data augmentation, specifically three techniques:

- Random Flip – Horizontal
- Random Rotation – 0.2
- Random Zoom – 0.2

This model also has the exact same architecture as the second one with the addition of the data augmentation method. As shown in figure 32, we can observe that the introduction of such regularization method with the addition of the dropout layers solves our overfitting problem.


*Figure 32*

As shown in figure 33, the accuracy of our model is 77 percent, which is of course considered to be a poor one. Therefore, we will try to increase it even more.

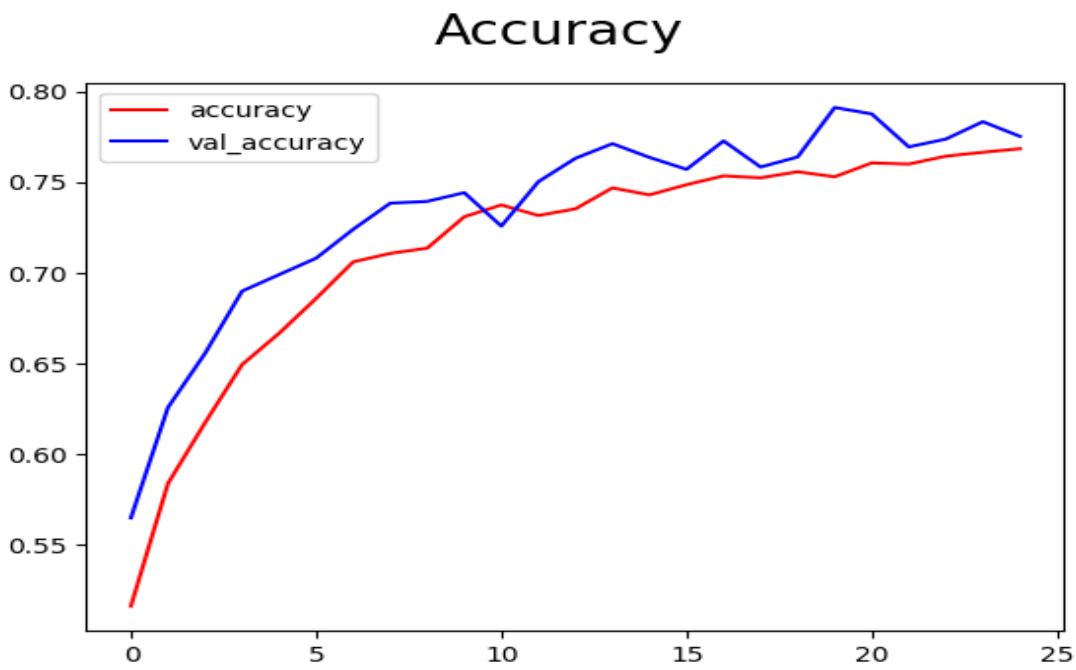


*Figure 33*

## iv. Fourth Model – Reducing Stride to 1:

The only adjustment we will be applying in this model is reducing the stride from 2 to 1. By observing figure 34, we can notice that our accuracy has increased significantly to 83 percent. Therefore, we can deduce that we have a very well-shaped architecture and it can be altered with the application of slight adjustments.

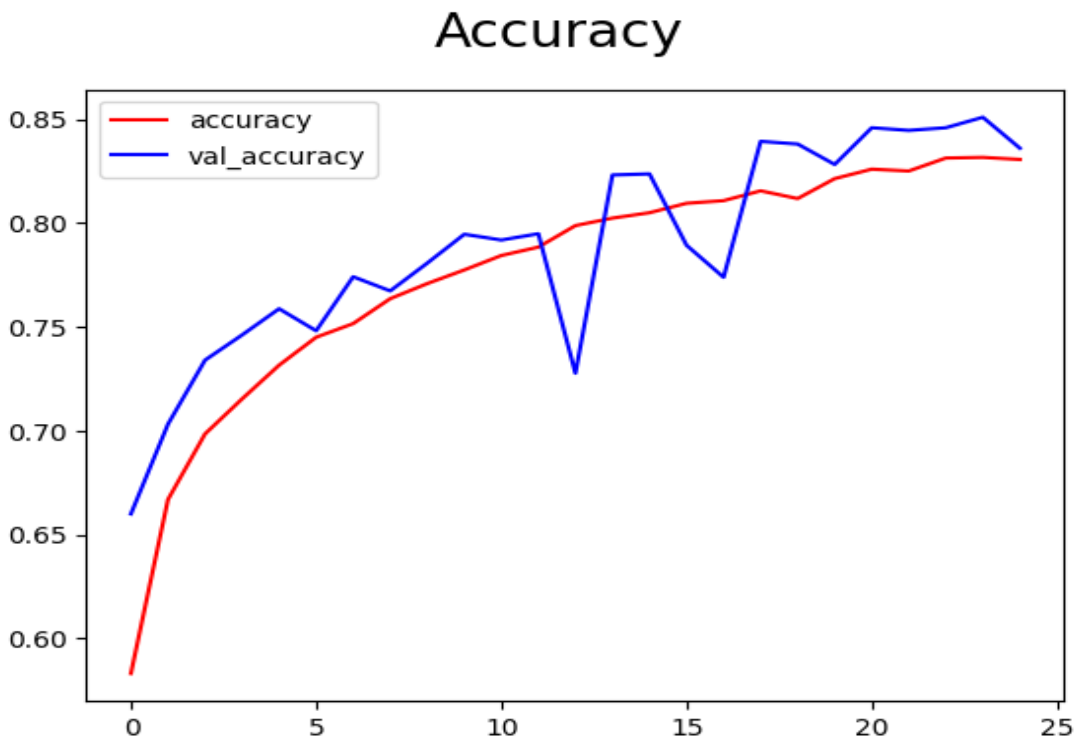


*Figure 34*

## v. Fifth Model – Reducing Image Size & Increasing Number Of Epochs:

Looking at the previous accuracy plots, we notice that our model doesn't reach plateau, hence the number of epochs is not enough and would need to be increased. However, deciding on the number of epochs can take several trials if we keep increasing it slightly and observe. Therefore, a good starting point would be doubling the number of epochs and adding 10 additional epochs as a 'buffer'.

Another thing we will be modifying is the image size, to try and reduce the image size to 64x64, the reason is in the previous model we had around 1.7 million parameters. Doing so reduces the number parameters to around 683,000 parameters.

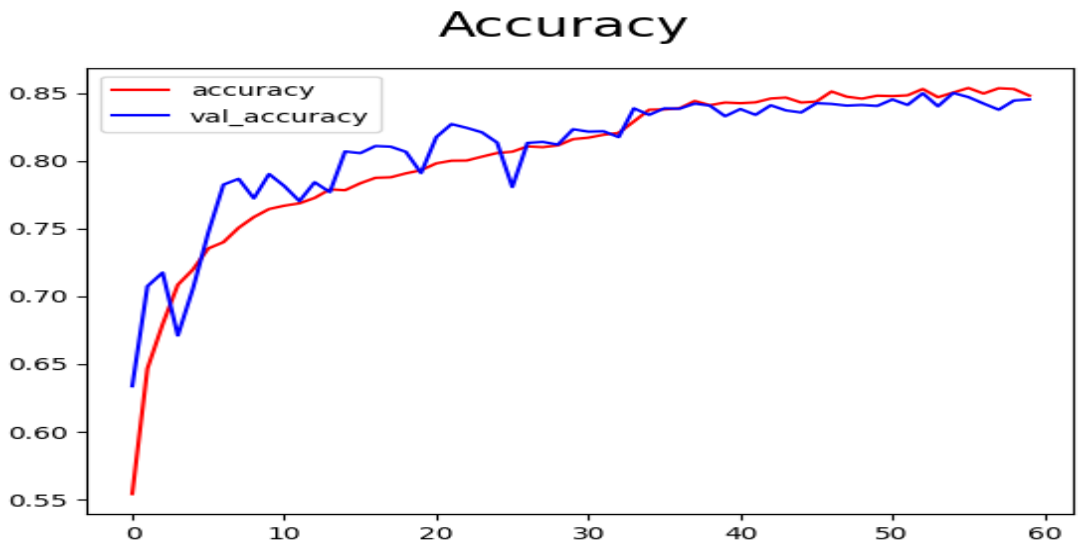


*Figure 35*

As shown in figure 35, we can clearly observe that our model does indeed reach plateau and we still maintain a very similar accuracy to our previous model, 84.5 percent accuracy.

## vi. Sixth Model – Increasing The Number Of Convolutional Layers:

As we discussed earlier, we want to observe whether the addition of an extra layer increases the accuracy and leads to a better model performance. Hence, we will add an additional convolutional layer with 256 as the number of filters. This actually led to a decrease in the number of parameters and a slight decrease in the accuracy, 83.75 percent accuracy, as shown in figure 36.

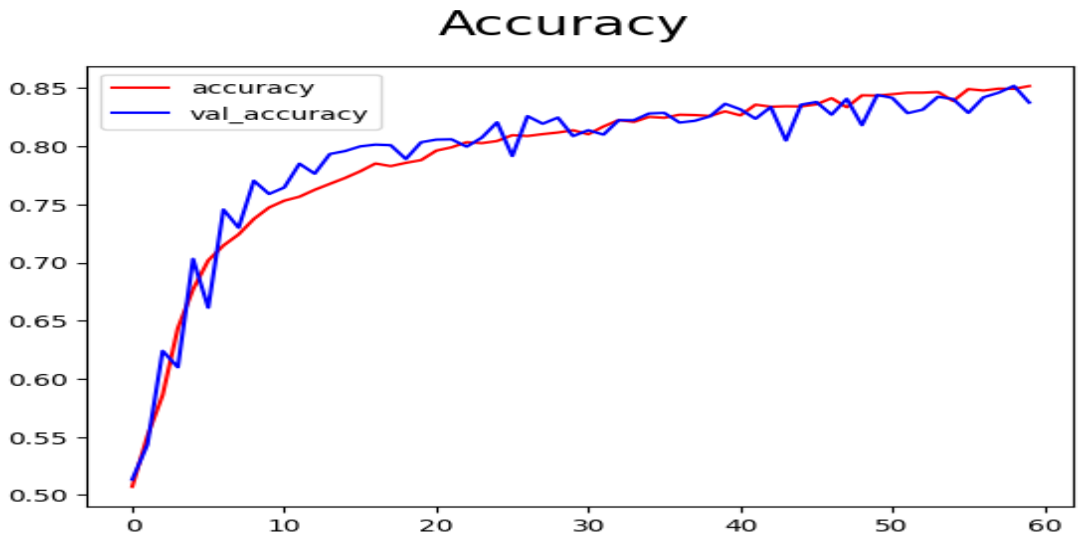


*Figure 36*

## vii. Seventh Model – Increasing The Image Size:

Our last trial before applying our final model was to observe whether having the initial image size in our model, 100x100, would significantly improve the performance of our model and lead to a noticeable increase in the model's accuracy. As shown in figure 37, we can clearly observe that our model does indeed perform a lot better and its accuracy is around 89.5 percent. It's also worth noting that the number of parameters decreased to around 912,000 parameters, compared to the fourth model. Hence, our final model that will be used is this current one with the application of 5-fold cross validation.

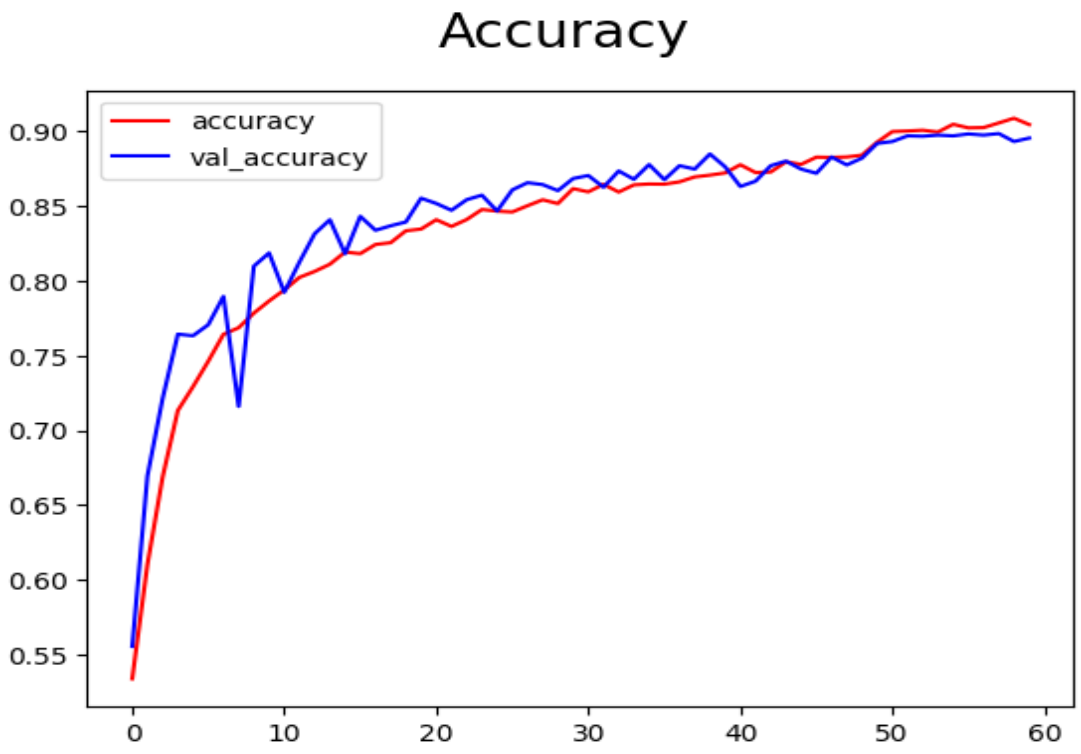


*Figure 37*

## viii. Final Model – Applying 5-Fold Cross-Validation:

Before implementing this model, we divided our training set into 5 folds, where each fold is composed of train set and validation set, 80/20, respectively,. Then, we implement 5-fold cross-validation, where in each fold we will keep the best performing model. Figures 38 & 39 are a representation of the accuracy & loss plots of each fold.

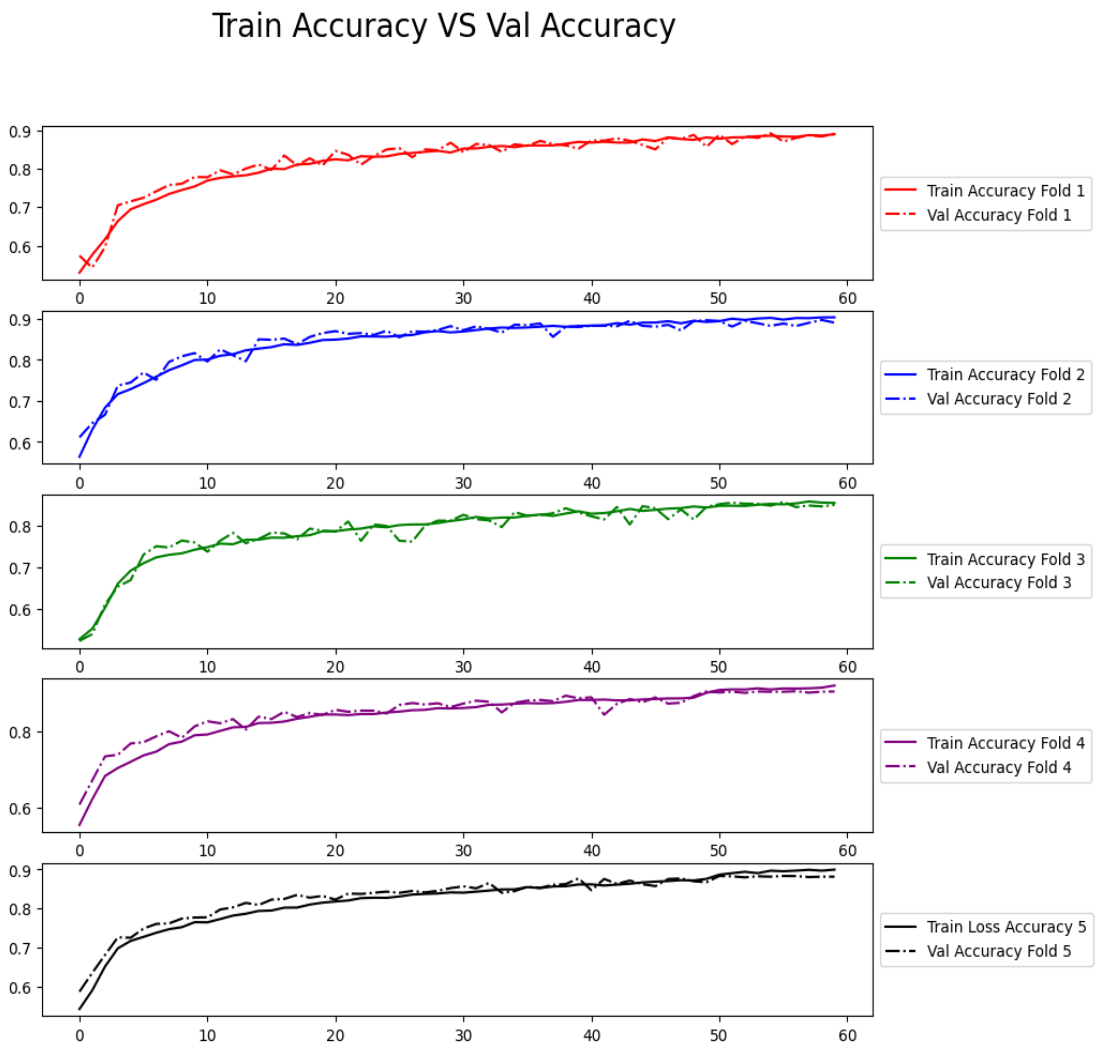


*Figure 38*

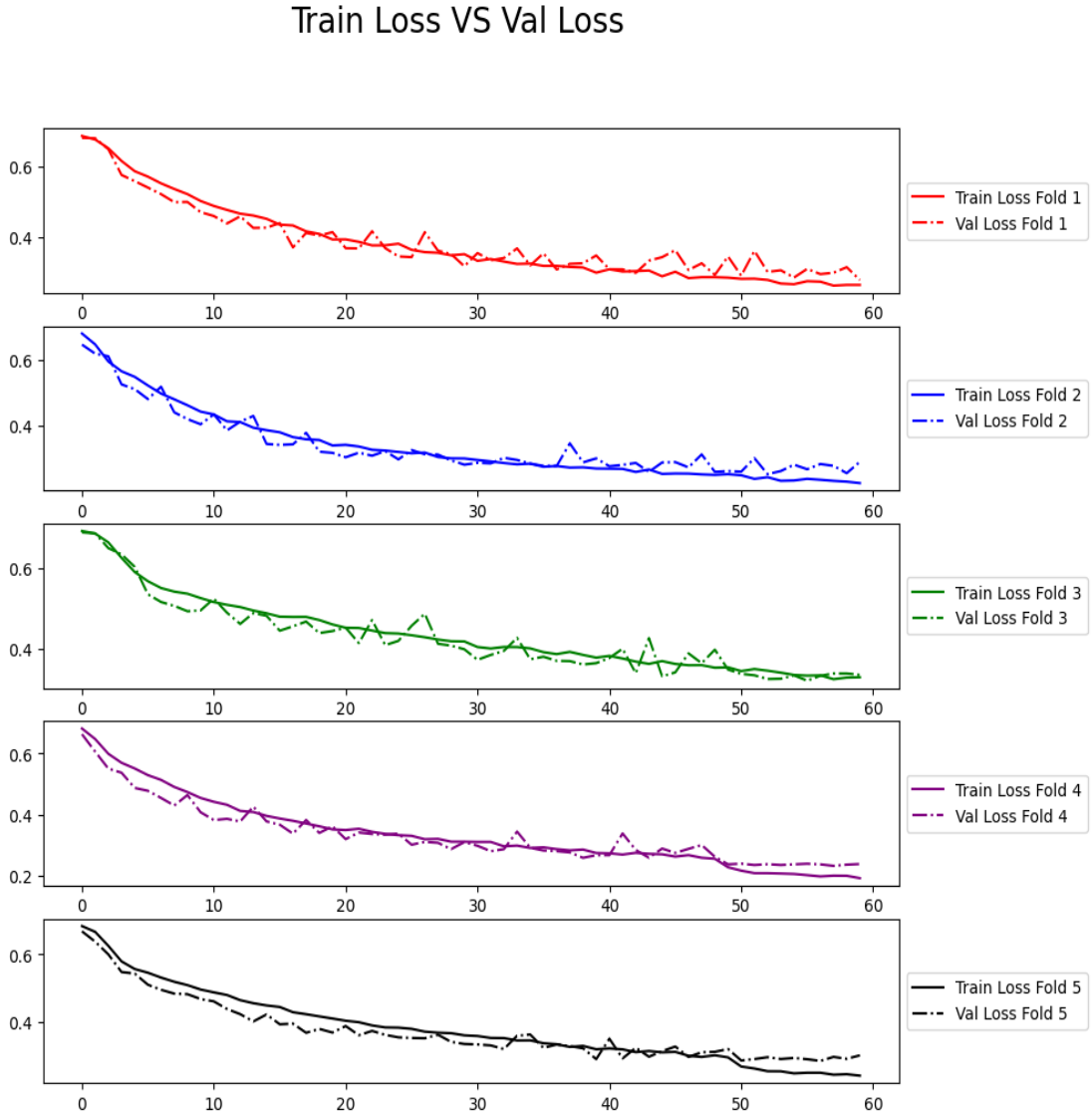


*Figure 39*

Finally, before evaluating our model on the test set, we can observe the average accuracy and loss scores of our model, shown in figure 40.

```
average loss 0.27686133682727815
average accuracy 0.8873673439025879
```

# 5. Model Evaluation:

After saving the best performing model of each fold and calculating the average accuracy of the model on the validation set, we then would want to evaluate these 5 best models on unseen data, test set. First, we evaluate each best model on the test and calculating 3 metrics which are:

- Precision: $^{TP}/_{TP + FP}$
- Recall: $^{TP}/_{TP + FN}$, also referred to as sensitivity.
- Accuracy, which is a measure of representing how many times the model is correct overall.

We will be focusing on the accuracy metric, since this was the measure that we have used when evaluating the performance of the model on the validation set. The following table is a representation of the accuracy of the best model of each fold on the test set.

| Model | Test Accuracy |
|---|---|
| **Best Model Fold 1** | 0.8817 |
| **Best Model Fold 2** | 0.8848 |
| **Best Model Fold 3** | 0.8804 |
| **Best Model Fold 4** | 0.8839 |
| **Best Model Fold 5** | 0.8860 |

The computed mean of these accuracy scores is the following:

| Average Test Accuracy |
|---|
| **0.8833** |

# 6. Conclusion:

In this study we were able to have a very good understanding of how a neural network operates, and how applying a convolutional neural network is much more efficient than the standard one.

Regarding the experimental part, we were able to observe the different types of CNN architectures and how to react when encountering an overfitting problem, introducing regularization methods. Another important note we were able to determine is that adding additional convolutional layers does in fact help reduce the number of parameters and improve the model's performance. It's worth noting that with the use of a GPU we can achieve much higher accuracy, if for instance we increase the image size, this was observed when we increased image size from 64x64 to 100x100.

Finally, as a last note, tuning certain hyperparameters have contributed to a significantly better performance, such as the stride hyperparameter. However, there are several other hyperparameters that can be altered and observed on how they would affect the model's performance, for instance batch size. This would require more extensive research with a more advanced machine, GPU.

# 7. Appendix:

## i. Final CNN Model 5-Fold CV:

```
def cnn_model():
    model = Sequential()
    #Resizing the images
    model.add(Resizing(100,100))
    # Scaling the image values between 0 & 1 instead of 0 to 255. This will help our Deep learning model to optimize faster and produce better results.
    model.add(Rescaling(1./255))
    #Data Augmentation to prevent overfitting
    model.add(RandomFlip('horizontal'))
    model.add(RandomRotation(0.2))
    model.add(RandomZoom(0.2))

    #Adding a convolutional layer and a MaxPooling layer
    #Each filter is going to be 3x3 in size and we are moving by 1 step/stride each time
    #Activation is going to be Rectified Linear Unit(relu), meaning any output below zero is going to be equal 0 and any other positive value is going to be preserved
    #MaxPooling helps us condense the information we get after the activation with the aim of getting the maximum values
    #MaxPooling halves the output of the Convolutional layer

    #First Layer
    model.add(Conv2D(32, (3,3), 1, activation = 'relu'))
    model.add(MaxPooling2D(2,2))

    #Second Layer
    model.add(Conv2D(64, (3,3), 1, activation = 'relu'))
    model.add(MaxPooling2D(2,2))

    #Third Layer
    model.add(Conv2D(128, (3,3), 1, activation = 'relu'))
    model.add(MaxPooling2D(2,2))

    #Fourth Layer
    model.add(Conv2D(256, (3,3), 1, activation = 'relu'))
    model.add(MaxPooling2D(2,2))

    model.add(Dropout(0.2))

    #We have condensed the rows and the width and the number of filters will form the channel value
    #Now the aim when flattening the data is condense the channel value to a single value
    model.add(Flatten())

    model.add(Dense(128, activation = 'relu'))

    model.add(Dropout(0.5))
    #Now we are condensing them even more to get the final value (0 or 1 --> cat or dog)
    model.add(Dense(1, activation = 'sigmoid'))

    #List of optimizers can be checked --> (tf.optimizers.)
    model.compile('adam', loss = tf.losses.BinaryCrossentropy(), metrics = ['accuracy'])
    return model
```

```
def fit_model(tr, val, iterator):
    model = None
    model = cnn_model()

    #define the model checkpoint callback -> this will keep on saving the model as a physical file
    model_checkpoint = ModelCheckpoint(models_dir+get_model_name(iterator), verbose=1, monitor='val_accuracy', mode=max, save_best_only=True)

    call_backs_list = [model_checkpoint]

    #fitting the model
    hist = model.fit(tr, validation_data = val, epochs=60, verbose = 1, callbacks = [call_backs_list, ReduceLROnPlateau()])

    #Evaluating the model
    best_model = load_model('saved_models/model_'+str(iterator)+'.h5')
    scores = best_model.evaluate(val, verbose = 0)
    cv_loss_scores.append(scores[0])
    cv_accuracy_scores.append(scores[1])
    iterator = iterator+1

    return hist
```

```
model_hist = []
for i in range(n_folds):
    train_data = tf.keras.utils.image_dataset_from_directory('Fold_'+str(i+1)+'/train', image_size = (100,100))
    val_data = tf.keras.utils.image_dataset_from_directory('Fold_'+str(i+1)+'/val', image_size = (100,100))
    print("Training On Fold: ", i+1)
    model_hist.append(fit_model(train_data, val_data, (i+1)))
    print("======="*10, end="\n\n\n")
```

| Layer (type) | Output Shape | Param # |
|---|---|---|
| resizing_16 (Resizing) | (None, 100, 100, 3) | 0 |
| rescaling_16 (Rescaling) | (None, 100, 100, 3) | 0 |
| random_flip_16 (RandomFlip) | (None, 100, 100, 3) | 0 |
| random_rotation_16 (RandomRotation) | (None, 100, 100, 3) | 0 |
| random_zoom_16 (RandomZoom) | (None, 100, 100, 3) | 0 |
| conv2d_58 (Conv2D) | (None, 98, 98, 32) | 896 |
| max_pooling2d_58 (MaxPooling2D) | (None, 49, 49, 32) | 0 |
| conv2d_59 (Conv2D) | (None, 47, 47, 64) | 18496 |
| max_pooling2d_59 (MaxPooling2D) | (None, 23, 23, 64) | 0 |
| conv2d_60 (Conv2D) | (None, 21, 21, 128) | 73856 |
| max_pooling2d_60 (MaxPooling2D) | (None, 10, 10, 128) | 0 |
| conv2d_61 (Conv2D) | (None, 8, 8, 256) | 295168 |
| max_pooling2d_61 (MaxPooling2D) | (None, 4, 4, 256) | 0 |
| dropout_32 (Dropout) | (None, 4, 4, 256) | 0 |
| flatten_16 (Flatten) | (None, 4096) | 0 |
| dense_32 (Dense) | (None, 128) | 524416 |
| dropout_33 (Dropout) | (None, 128) | 0 |
| dense_33 (Dense) | (None, 1) | 129 |

Total params: 912,961
Trainable params: 912,961
Non-trainable params: 0

# 8. References:


i. Sultana, Farhana, et al. “Advancements in Image Classification Using Convolutional Neural Network.” 2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN), 2018

ii. Alzubaidi, Laith, et al. “Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions.” Journal of Big Data, vol. 8, no. 1, 2021

iii. Sharma-, ByPalash, et al. “7 Popular Image Classification Models in ImageNet Challenge (ILSVRC) Competition History.” MLK - Machine Learning Knowledge, 17 Oct. 2020

iv. DeepAI. “Neural Network.” DeepAI, DeepAI, 17 May 2019

v. Godoy, Daniel. “Understanding Binary Cross-Entropy / Log Loss: A Visual Explanation.” Medium, Towards Data Science, 10 July 2022

vi. Yathish, Vishal. “Loss Functions and Their Use in Neural Networks.” Medium, Towards Data Science, 4 Aug. 2022

vii. Udofia, Udeme. “Basic Overview of Convolutional Neural Network (CNN).” Medium, DataSeries, 20 Sept. 2019

viii. DeepAI. “Stride (Machine Learning).” DeepAI, DeepAI, 17 May 2019

ix. Biswal, Avijeet. “Convolutional Neural Network Tutorial [Update].” Simplilearn.com, Simplilearn, 16 Feb. 2023

x. Budhiraja, Amar. “Learning Less to Learn Better-Dropout in (Deep) Machine Learning.” Medium, Medium, 6 Mar. 2018

xi. Maklin, Cory. “Dropout Neural Network Layer in Keras Explained.” Medium, Towards Data Science, 9 May 2022

xii. Yadav, Harsh. “Dropout in Neural Networks.” Medium, Towards Data Science, 5 July 2022

xiii. baeldung, Written by: “How Relu and Dropout Layers Work in Cnns.” Baeldung on Computer Science, 25 Nov. 2022

xiv. Dumane, Govinda. “Introduction to Convolutional Neural Network (CNN) Using Tensorflow.” Medium, Towards Data Science, 19 Mar. 2020