IMAGE CLASSIFICATION

(DOGS AND CATS)

Using Convolutional Neural Networks (CNNs)

Project Report

Prepared By -

Name	Sudhanshu Kakkar
CWID	20036779
Class / Subject	2025S EE 628-WS1

OBJECTIVE

The main goal of this project is to build a model that can tell the difference between cats and dogs just by looking at images. To do this, I used a Convolutional Neural Network (CNN), which is good at understanding pictures. The model was trained on a dataset with labelled images of cats and dogs so it could learn the patterns that make them look different—like shapes, edges, and textures. After training, I tested the model on the validation data set to see how well it performs, using graphs for accuracy and loss, as well as a confusion matrix and classification report to better understand its predictions.

Data Set Used

A total of 25,000 labelled images of cats and dogs from Kaggle.com were used for model building and evaluation.
80% of the images were used as the training dataset to train the model, while the remaining 20% were used as the validation dataset to evaluate its performance.

WHY CNN ?

CNNs (Convolutional Neural Networks) are one of the best options for image classification tasks like cats vs. dogs. They're designed specifically for working with image data.

The main thing that makes CNNs powerful is their use of **filters**. These filters move across the image and pick up key features like **edges, textures, shapes,** and more. Instead of just using raw pixel values, CNNs learn patterns from small sections of the image, layer by layer.

Also, one big advantage is that **CNNs maintain the spatial structure of the image**. This means the position and relationship between pixels are not lost, because unlike regular neural networks, CNNs don't flatten the input image right at the start. This helps the model understand where features are located in the image, which is super important for tasks like recognizing a cat's face vs. a dog's body.

All of this makes CNNs faster, more efficient, and much better suited for working with images than regular fully connected neural networks.

WHY NOT FNN?

Fully Connected Neural Networks (FNNs) aren't the best choice for image classification because they don't keep the 2D structure of images. When we pass an image to an FNN, it gets flattened into a long list of numbers, and that removes important spatial information—like where things are located in the image. For example, if a cat is in the top left corner or the bottom right, the FNN might treat both the same way and struggle to recognize it. This loss of position and structure makes FNNs less effective for working with image data compared to CNNs.

BUILDING THE MODEL

APPROACH 1

Below configuration we have used -

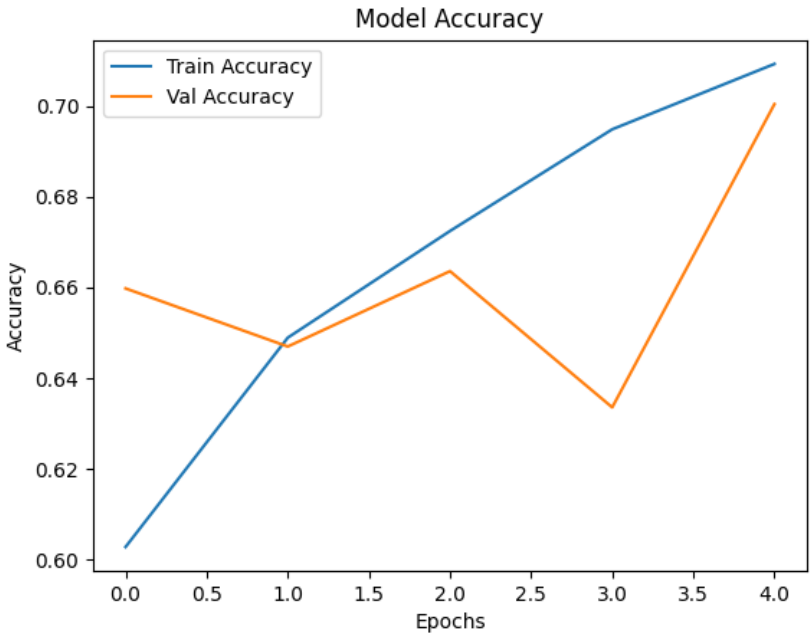
No of epochs	5
No. Convolution Layers with MaxPooling Layer	1

In the below code **epochs** is highlighted–

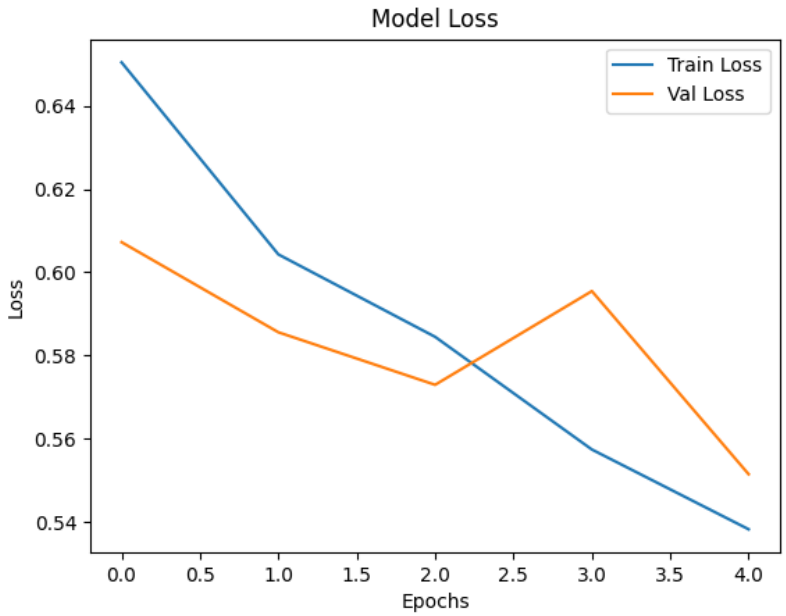
```
history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs= 5
)
```

In the below code **Convolution Layers with MaxPooling Layer** is highlighted in red –

```
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(2,2),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(1)
])
```



- Training Accuracy increases steadily from ~60% to ~71%, indicating that the model is learning the patterns in the training dataset effectively.
- Validation Accuracy shows fluctuations: it starts at ~66%, dips slightly, then rises again to end near 70%.
- The gap between training and validation accuracy is not substantial, suggesting no severe overfitting



- Training Loss consistently decreases from ~0.65 to ~0.54, showing that the model is effectively minimizing error on the training data.
- Validation Loss also trends downward overall, though it exhibits a slight increase at epoch 3. This fluctuation could be due to temporary overfitting or batch variability.
- The final validation loss (~0.55) closely aligns with training loss, suggesting a good fit and no major overfitting.

APPROACH 2

Below configuration we have used -

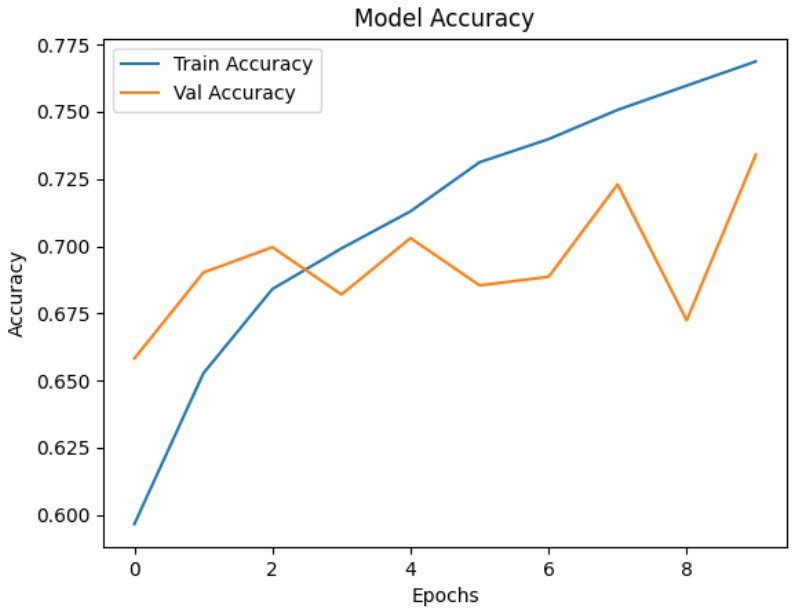
No of epochs	10
No. Convolution Layers with MaxPooling Layer	1

In the below code **epochs** is highlighted–

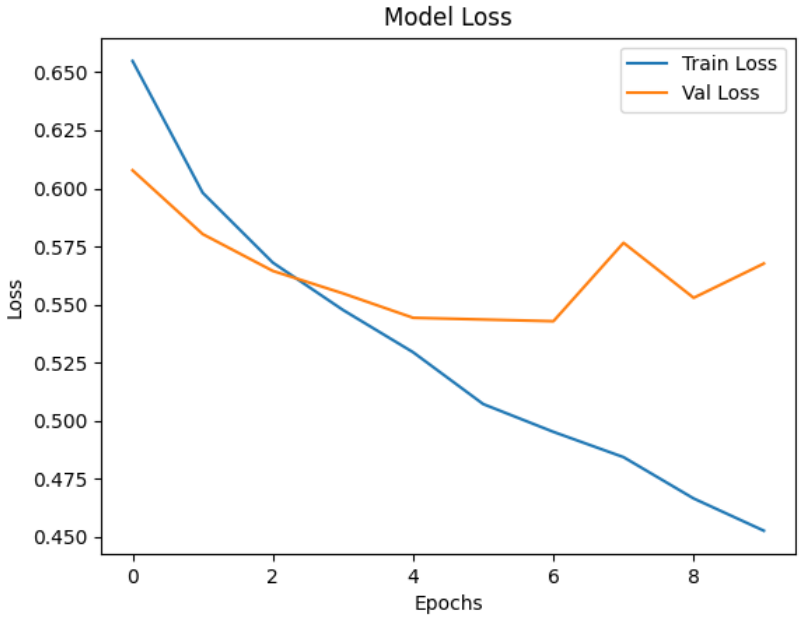
```
history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs= 10
)
```

In the below code **Convolution Layers with MaxPooling Layer** is highlighted in red –

```
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(2,2),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(1)  # Logits
])
```



- Training Accuracy shows a steady and consistent increase, rising from ~60% to ~77%. This indicates the model is effectively learning from the training dataset over time.
- Validation Accuracy fluctuates across epochs, peaking around epoch 7 (~72%) and ending close to 73%. These oscillations suggest that while the model is improving, the generalization to unseen data isn't perfectly stable.
- The gap between training and validation accuracy widens slightly in the later epochs, potentially hinting at mild overfitting. However, since the validation accuracy also improves by the end, this is not a major concern.



- Training Loss consistently decreases from ~0.65 to ~0.45, which is a strong indicator that the model is learning effectively and reducing error on the training set.
- Validation Loss initially decreases and then fluctuates after epoch 6, suggesting the model may be beginning to overfit slightly—learning patterns specific to the training data that don't generalize as well to unseen data.
- The increasing variance in validation loss while training loss continues to fall is a common sign that the model might benefit from early stopping, dropout layers, or more data.

APPROACH 3

Below configuration we have used -

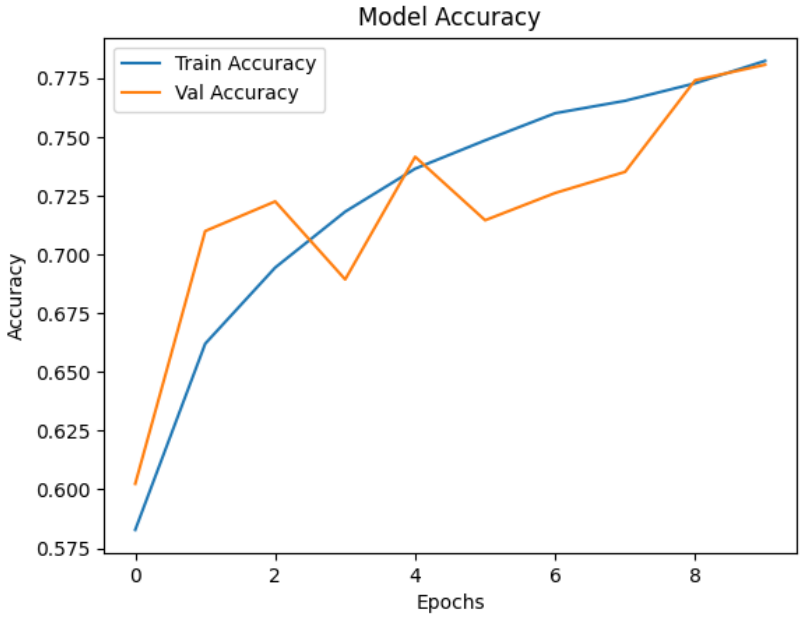
No of epochs	10
No. Convolution Layers with MaxPooling Layer	3

In the below code **epochs** is highlighted–

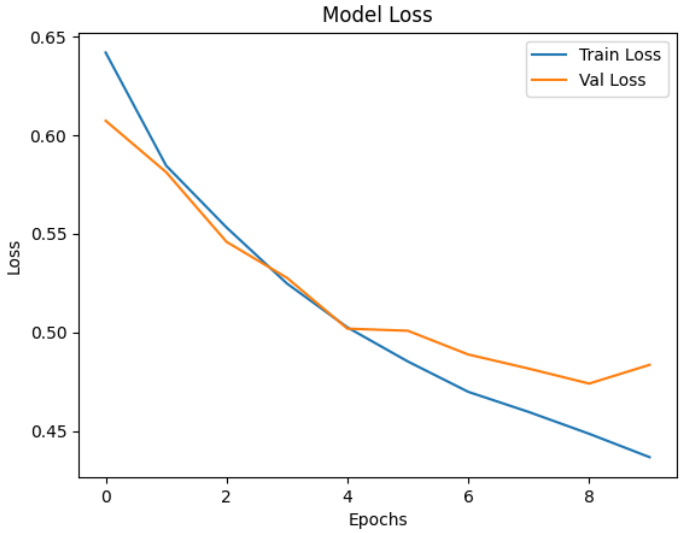
```
history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs= 10
)
```

In the below code **Convolution Layers with MaxPooling Layer** is highlighted –

```
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(2,2),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    Conv2D(128, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(1)
])
```



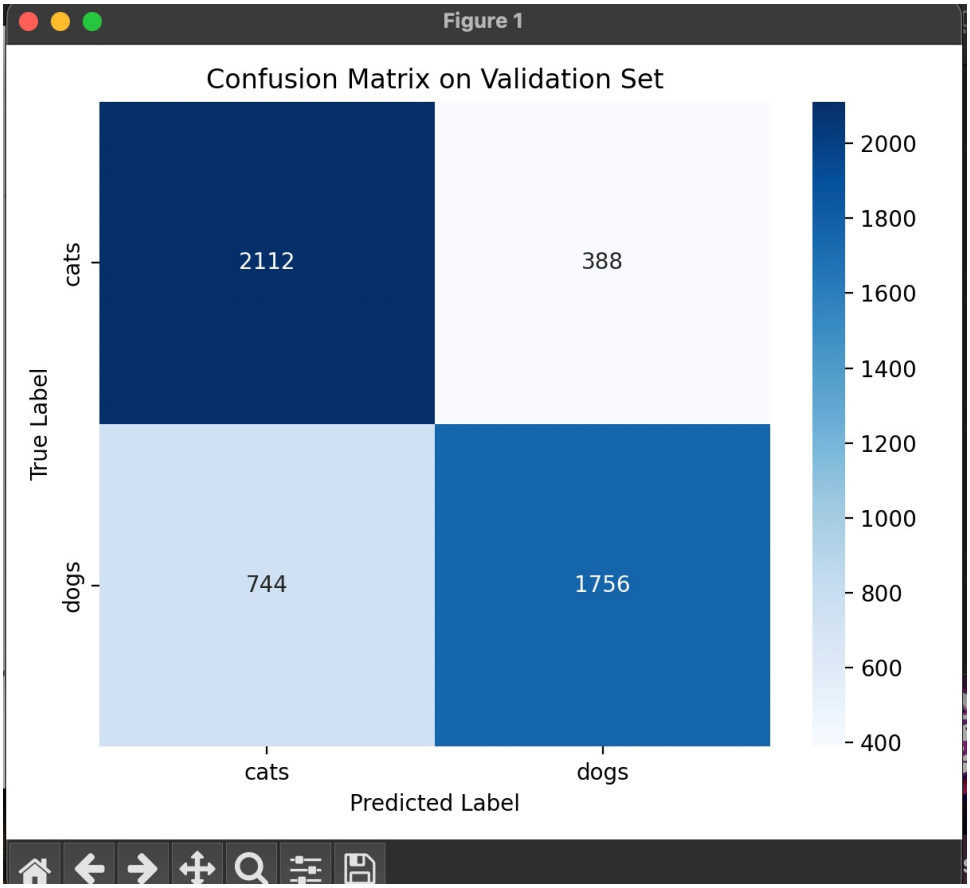
- Training Accuracy shows a steady and smooth increase, rising from ~58% to nearly 78%, indicating strong learning progression.
- Validation Accuracy also follows a generally increasing trend, ending close to the training accuracy. This convergence is an excellent sign, suggesting the model is well-generalized.
- Although the validation curve has minor fluctuations around epochs 3–6, the final epochs show minimal divergence between the two curves, demonstrating balanced learning without overfitting.
- This is likely the best-performing model among your trials, as it maintains high validation accuracy and smooth convergence.



- Training Loss consistently decreases from ~0.64 to ~0.43, indicating continuous improvement and reduced error during learning.
- Validation Loss mirrors this trend, steadily decreasing and closely following the training curve throughout.
- Around epoch 4, the training and validation loss curves nearly intersect and continue in sync—this is a strong indicator of excellent generalization and model stability.
- Minimal fluctuations in validation loss and tight alignment with training loss reflect no overfitting and suggest the model is likely at or near its optimal configuration.

CONFUSION MATRIX

After implementing the 3 approaches mentioned above, and analysing the accuracy and loss for each approach, I am finalizing the ***approach 3 with 10 epochs and 3 convolution layers and corresponding MaxPooling layers***.



This confusion matrix shows the model's performance on the validation set:

- The model correctly predicted **2,112 cat images** and **1,756 dog images**.
- It wrongly predicted **388 cats as dogs** and **744 dogs as cats**.
- Overall, the model is **better at identifying cats** than dogs, as the number of misclassified dogs is higher.
- Despite some misclassifications, the accuracy is still quite good, with most predictions being correct.

ACCURACY

From the confusion matrix obtained, accuracy of the model comes out to be 77.36%.

$$\text{Accuracy} = \frac{TP + TN}{Total} = \frac{2112 + 1756}{2112 + 388 + 744 + 1756} = \frac{3868}{5000} = 0.7736 \approx \boxed{77.36\%}$$

CLASSIFICATION REPORT

Classification Report:

	precision	recall	f1-score	support
cats	0.74	0.84	0.79	2500
dogs	0.82	0.70	0.76	2500
accuracy			0.77	5000
macro avg	0.78	0.77	0.77	5000
weighted avg	0.78	0.77	0.77	5000

The model has an overall **accuracy of 77%**

For **cats**:

- It finds most of the actual cat images (**recall = 84%**).
- But sometimes it guesses cat when it's not (**precision = 74%**).

For **dogs**:

- It's good at guessing dogs correctly (**precision = 82%**),
- But it misses more real dog images compared to cats (**recall = 70%**).