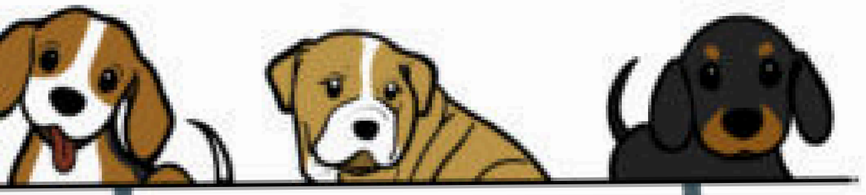
GEL521 - 202420
Machine Learning

Cats and Dogs

Presented To: Dr Hayssam SERHAN
Presented By: Antonio HADDAD 202200238
Elias-Charbel SALAMEH 202201047

Background of the Study

We will use the kagglecatsanddogs_5340 dataset in our Convolutional Neural Network to train our machine to be able to classify images of **cats** and **dogs** correctly.



OBJECT RECOGNITION ALGORITHM

CAT

DOG

Why use a CNN?

What is a CNN?

Well suited for tasks such as **image recognition** and **classification**

Why not use our basic Neural network?

- Computanional Efficiency
- Parameter Sharing
- Spatial Hiearchies

How does a CNN work?

- Convolutional Layers
- Pooling Layers
- Activation functions
- Fully Connected Layers

Objectives

High accuracy

Train and predict the available data accurately

Avoid Overfitting

The machine may memorize the patterns instead

Methodology

import the KERAS library

Data pre-processing

Training

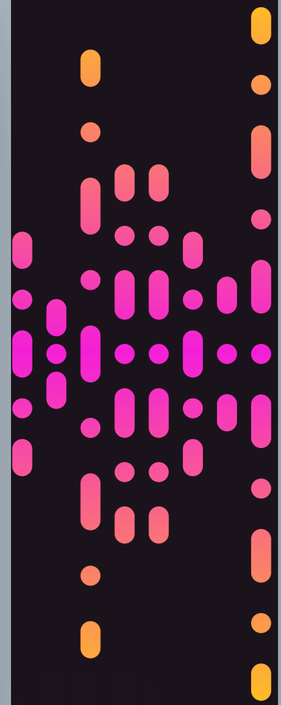
Plotting and testing

Increase accuracy

Predict new images

Setup Stage

```
split_dataset/
├── train/
│   ├── Cat/
│   │   ├── image1.jpg
│   │   ├── image2.jpg
│   │   └── ...
│   └── Dog/
│       ├── image1.jpg
│       ├── image2.jpg
│       └── ...
└── test/
    ├── Cat/
    │   ├── image1.jpg
    │   ├── image2.jpg
    │   └── ...
    └── Dog/
        ├── image1.jpg
        ├── image2.jpg
        └── ...
```



```python
import os
import shutil
import random

# Define the source directory containing the classes "Cat" and "Dog"
source_directory = "dataset/"

# Define the destination directory where the split data will be stored
destination_directory = "split_dataset/"

# Define the ratio for splitting (80% training, 20% testing)
split_ratio = 0.8

# Create the destination directory if it doesn't exist
if not os.path.exists(destination_directory):
    os.makedirs(destination_directory)

# Function to count the number of files in a directory
def count_files(directory):
    return sum(len(files) for _, _, files in os.walk(directory))

# Iterate through each class directory ("Cat" and "Dog") in the source directory
for class_name in os.listdir(source_directory):
    class_directory = os.path.join(source_directory, class_name)

    # Create subdirectories for training and testing
    train_directory = os.path.join(destination_directory, "train", class_name)
    test_directory = os.path.join(destination_directory, "test", class_name)
    os.makedirs(train_directory, exist_ok=True)
    os.makedirs(test_directory, exist_ok=True)

    # Get a list of image files in the class directory
    image_files = [f for f in os.listdir(class_directory) if os.path.isfile(os.path.join(class_directory, f))]

    # Shuffle the list of image files randomly
    random.shuffle(image_files)

    # Calculate the number of images for training and testing based on the split ratio
    num_train_images = int(len(image_files) * split_ratio)
    num_test_images = len(image_files) - num_train_images

    # Split the image files into training and testing sets
    train_images = image_files[:num_train_images]
    test_images = image_files[num_train_images:]

    # Move training images to the training directory
    for image in train_images:
        source_path = os.path.join(class_directory, image)
        destination_path = os.path.join(train_directory, image)
        shutil.copy(source_path, destination_path)

    # Move testing images to the testing directory
    for image in test_images:
        source_path = os.path.join(class_directory, image)
        destination_path = os.path.join(test_directory, image)
        shutil.copy(source_path, destination_path)

# Display the length of each directory
print("Number of images in training directory (Cat):", count_files(os.path.join(destination_directory, "train", "Cat")))
print("Number of images in training directory (Dog):", count_files(os.path.join(destination_directory, "train", "Dog")))
print("Number of images in testing directory (Cat):", count_files(os.path.join(destination_directory, "test", "Cat")))
print("Number of images in testing directory (Dog):", count_files(os.path.join(destination_directory, "test", "Dog")))
```

Setup Stage

```python
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import random
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator # type: ignore
from tensorflow.keras.models import Sequential # type: ignore
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense # type: ignore
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, LearningRateScheduler # type: ignore

def display_images_with_labels(generator, model, num_images, rows, columns):
        # Get the class labels
        class_labels = list(generator.class_indices.keys())

        # Get a batch of images and their true labels from the generator
        images, true_labels = next(generator)

        # Get predictions for the images
        predictions = model.predict(images)
        predicted_labels = [class_labels[np.argmax(pred)] for pred in predictions]

        # Calculate accuracy
        true_indices = np.argmax(true_labels, axis=1)
        accuracy = np.mean(np.equal(true_indices, np.argmax(predictions, axis=1)))

        # Randomly select num_images indices
        indices = random.sample(range(len(images)), num_images)

        # Display the images and labels
        fig, axes = plt.subplots(rows, columns, figsize=(15, 15))

        for i, ax in enumerate(axes.flat):
                ax.imshow(images[indices[i]])
                true_label = class_labels[np.argmax(true_labels[indices[i]])]
                predicted_label = predicted_labels[indices[i]]

                # Check if predicted label matches true label
                if true_label == predicted_label:
                        title_color = 'green'
                else:
                        title_color = 'red'

                ax.set_title(f'True Label: {true_label}\nPredicted Label: {predicted_label}', color=title_color)
                ax.axis('off')

        plt.tight_layout()
        plt.show()

        print("Accuracy:", accuracy)

print("Setup Done")
```

```python
# Data generators for train and test data
train_data_generator = ImageDataGenerator(rescale=1./255)

test_data_generator = ImageDataGenerator(rescale=1./255)

# Load and preprocess train and test data
train_generator = train_data_generator.flow_from_directory(
        'split_dataset/train/',
        target_size=(256, 256),
        batch_size=64,
        class_mode='categorical',
        shuffle=True)

test_generator = test_data_generator.flow_from_directory(
        'split_dataset/test/',
        target_size=(256, 256),
        batch_size=64,
        class_mode='categorical',
        shuffle=True)
```

```python
# Model architecture
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 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(2, activation='softmax')
])
```

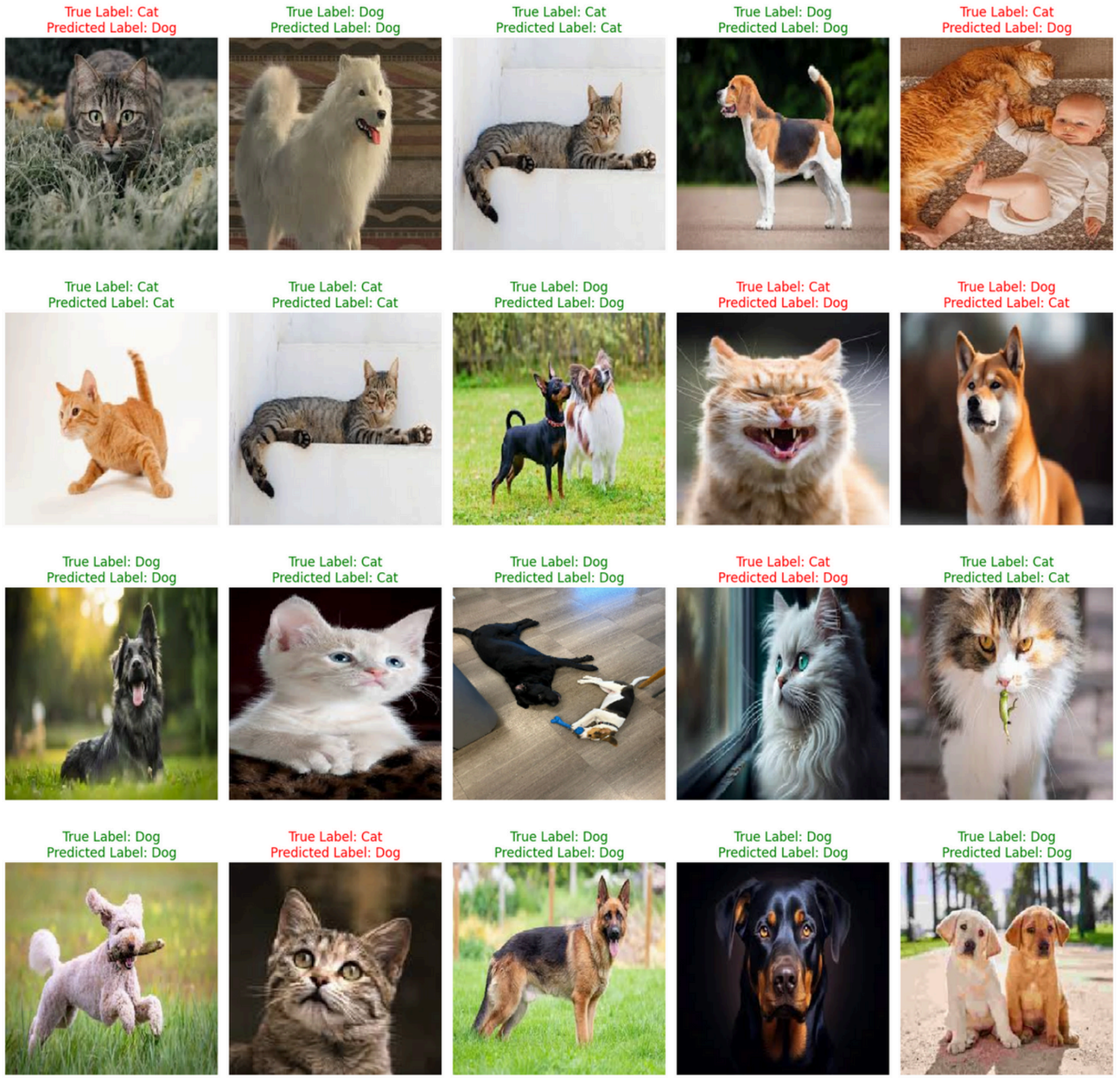
size=(256,256);
batch_size=64;
epochs=10

```
Found 24998 images belonging to 2 classes.
Found 24998 images belonging to 2 classes.
Epoch 1/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 530s 1s/step - accuracy: 0.6152 - loss: 0.8811 - val_accuracy: 0.7661 - val_loss: 0.4852
Epoch 2/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 408s 1s/step - accuracy: 0.7800 - loss: 0.4572 - val_accuracy: 0.8555 - val_loss: 0.3336
Epoch 3/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 408s 1s/step - accuracy: 0.8517 - loss: 0.3387 - val_accuracy: 0.9232 - val_loss: 0.2359
Epoch 4/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 407s 1s/step - accuracy: 0.9204 - loss: 0.2075 - val_accuracy: 0.9706 - val_loss: 0.1069
Epoch 5/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 410s 1s/step - accuracy: 0.9730 - loss: 0.0773 - val_accuracy: 0.9944 - val_loss: 0.0297
Epoch 6/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 413s 1s/step - accuracy: 0.9906 - loss: 0.0309 - val_accuracy: 0.9977 - val_loss: 0.0103
Epoch 7/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 414s 1s/step - accuracy: 0.9933 - loss: 0.0215 - val_accuracy: 0.9934 - val_loss: 0.0298
Epoch 8/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 432s 1s/step - accuracy: 0.9953 - loss: 0.0188 - val_accuracy: 0.9984 - val_loss: 0.0097
Epoch 9/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 434s 1s/step - accuracy: 0.9955 - loss: 0.0155 - val_accuracy: 0.9984 - val_loss: 0.0063
Epoch 10/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 428s 1s/step - accuracy: 0.9966 - loss: 0.0116 - val_accuracy: 0.9936 - val_loss: 0.0211
391/391 ━━━━━━━━━━━━━━━━━━━━ 89s 227ms/step - accuracy: 0.9934 - loss: 0.0223
Test Loss: 0.021131806075572968
Test Accuracy: 0.9935594797134399
391/391 ━━━━━━━━━━━━━━━━━━━━ 85s 216ms/step
```

Training accuracy = 99.34%

Testing accuracy = 99.36%

test_accuracy
=
70%

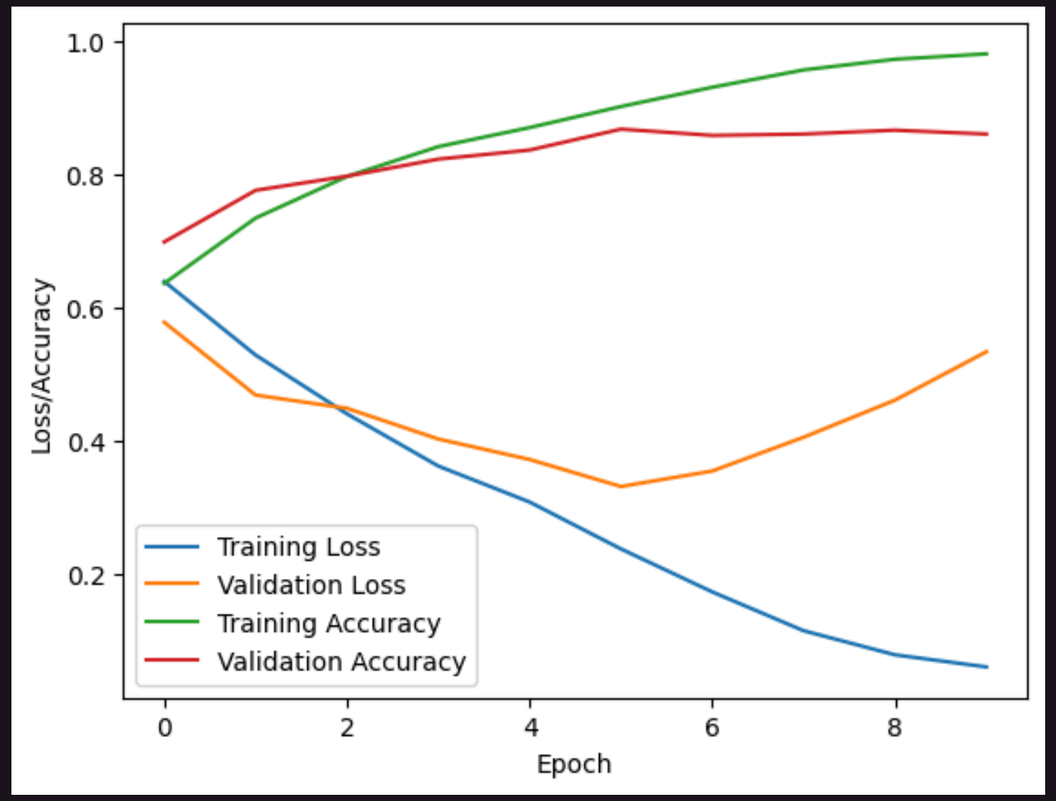


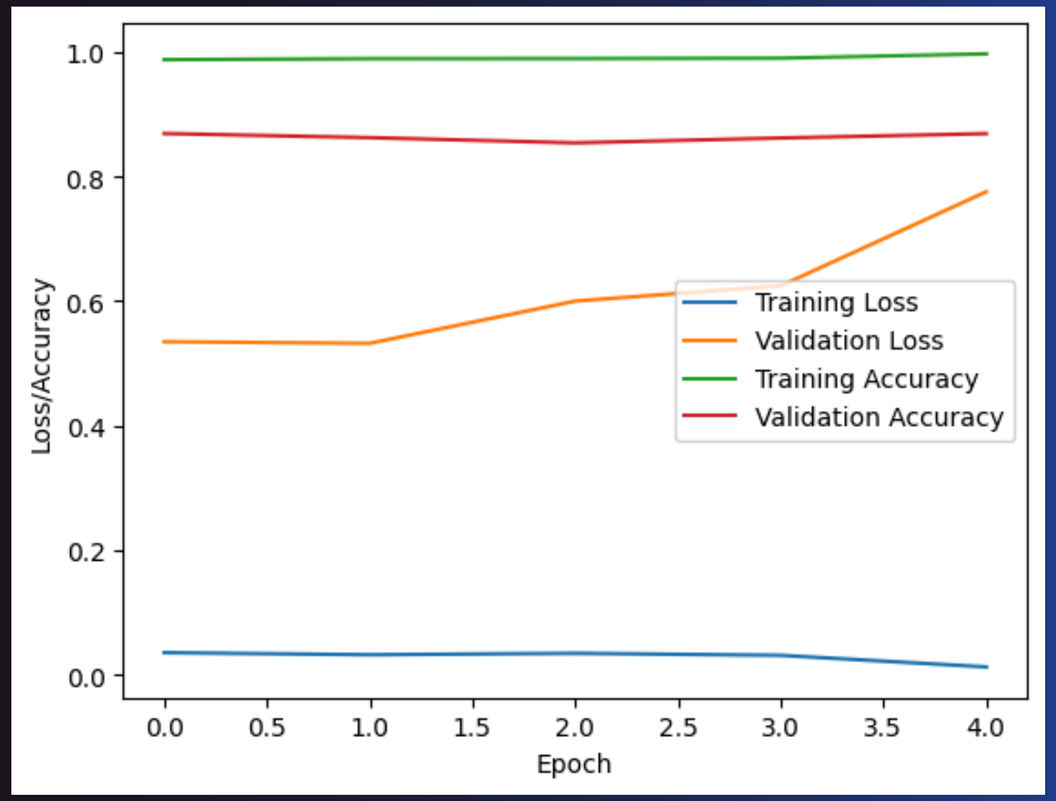
This is an example of overfitting.

What is Overfitting ?

Overfitting occurs when a model learns to perform well on the training data but **fails to generalize** to unseen data.

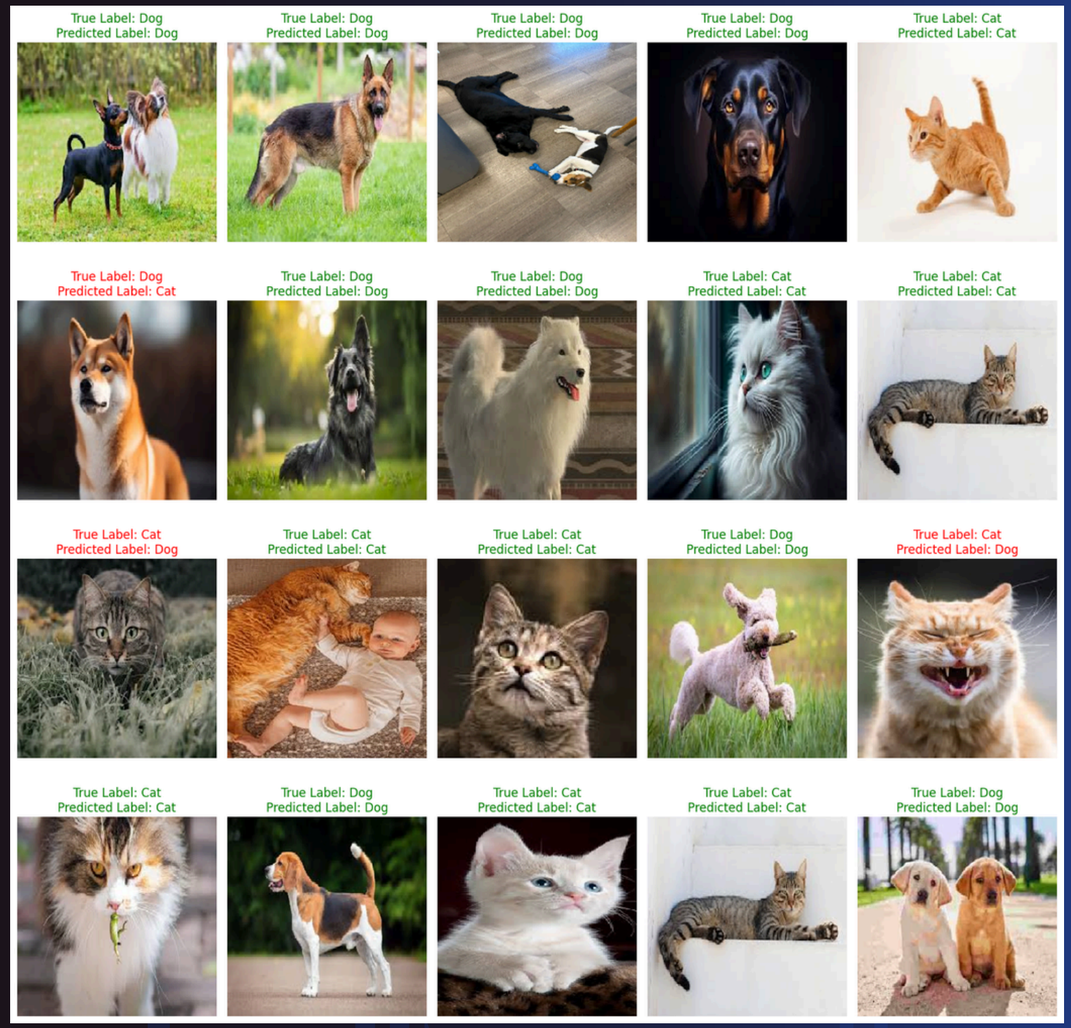
Adding a dropout layer without augmentation.





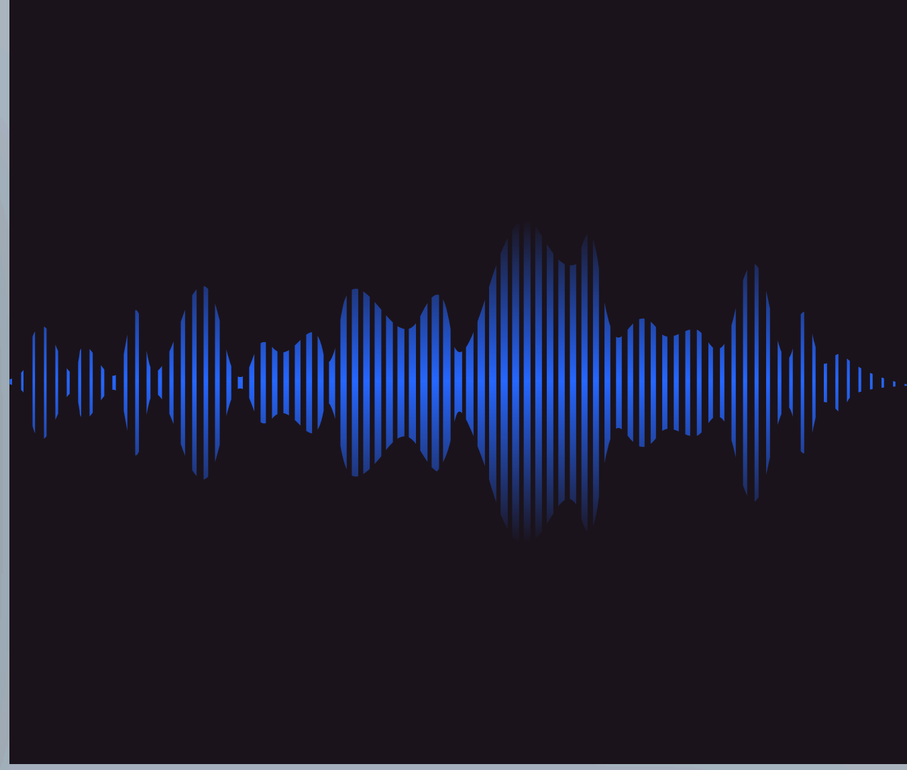
Training accuracy = 99.73%

Another case of Overfitting



Testing accuracy = 84.375% Validation accuracy = 85%

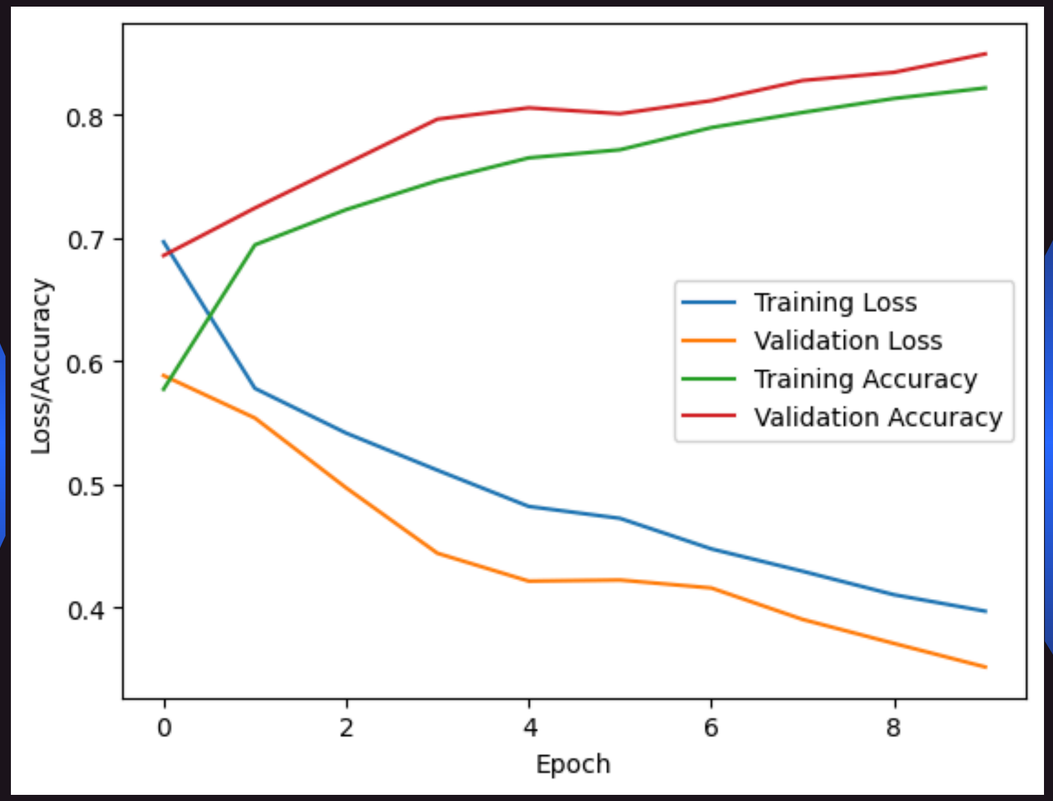
```python
# Data generators for train and test data
train_data_generator = ImageDataGenerator(rescale=1./255,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)
```



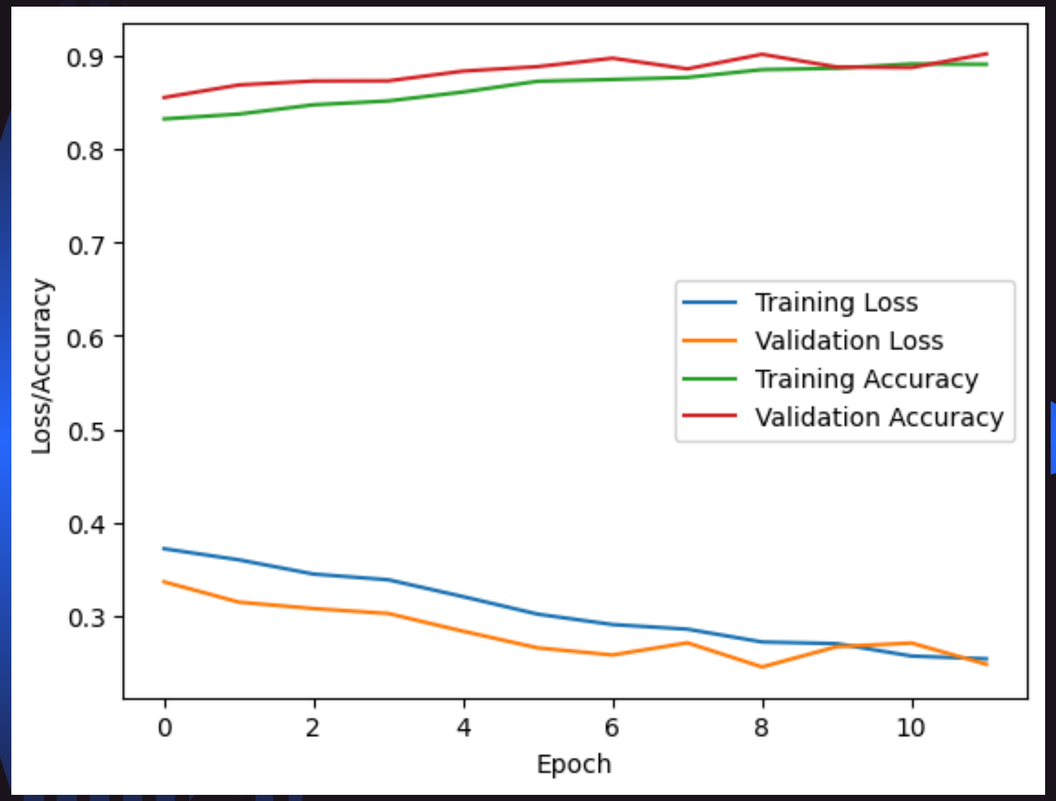
```python
# Define callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_model.keras', monitor='val_loss', save_best_only=True)
learning_rate_scheduler = LearningRateScheduler(lambda epoch, lr: lr * 0.9 if epoch % 5 == 0 else lr)

# Train the model with callbacks
history = model.fit(train_generator, epochs=20, validation_data=test_generator,
                    callbacks=[early_stopping, model_checkpoint, learning_rate_scheduler])
```

size=(200,200); batch_size=64; epochs=10 with augmentation

during training

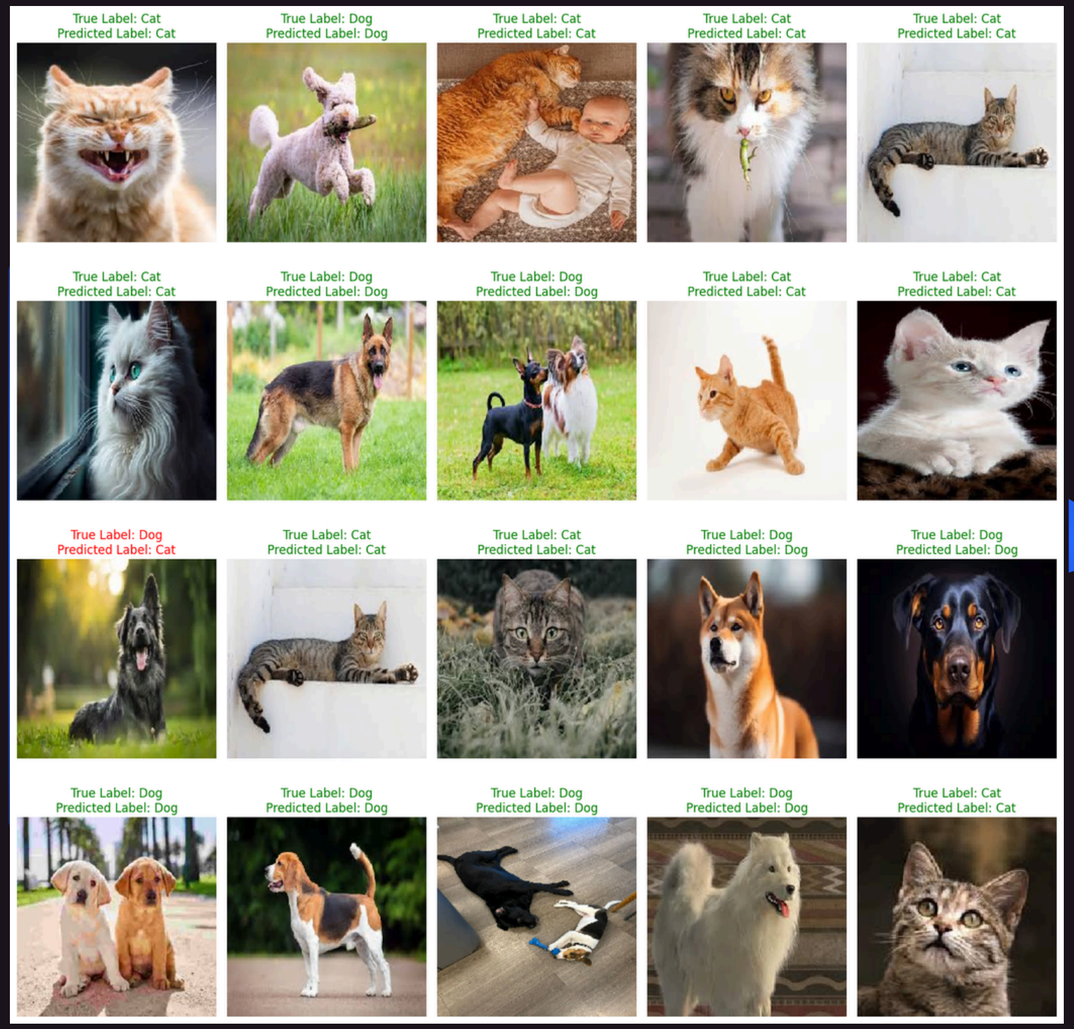


during callbacks



Training accuracy = 88.92%
Testing accuracy = 89.0625%
Validation accuracy = 95%

size=(200,200); batch_size=64; epochs=10 with augmentation



We add 2 convolution layers and 2 MaxPooling layers

```python
# Model architecture
model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(200, 200, 3)),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(512, activation='relu'),
        Dense(2, activation='softmax')
])
```

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 198, 198, 32)	896
max_pooling2d_3 (MaxPooling2D)	(None, 99, 99, 32)	0
conv2d_4 (Conv2D)	(None, 97, 97, 64)	18,496
max_pooling2d_4 (MaxPooling2D)	(None, 48, 48, 64)	0
conv2d_5 (Conv2D)	(None, 46, 46, 128)	73,856
max_pooling2d_5 (MaxPooling2D)	(None, 23, 23, 128)	0
conv2d_6 (Conv2D)	(None, 21, 21, 128)	147,584
max_pooling2d_6 (MaxPooling2D)	(None, 10, 10, 128)	0
conv2d_7 (Conv2D)	(None, 8, 8, 128)	147,584
max_pooling2d_7 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten_1 (Flatten)	(None, 2048)	0
dense_2 (Dense)	(None, 512)	1,049,088
dense_3 (Dense)	(None, 2)	1,026

Total params: 4,315,592 (16.46 MB)

Before callbacks

```
Epoch 1/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 176s 552ms/step - accuracy: 0.5203 - loss: 0.6911 - val_accuracy: 0.6287 - val_loss: 0.6421
Epoch 2/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 172s 543ms/step - accuracy: 0.6286 - loss: 0.6469 - val_accuracy: 0.6965 - val_loss: 0.6155
Epoch 3/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 172s 543ms/step - accuracy: 0.6765 - loss: 0.6034 - val_accuracy: 0.7566 - val_loss: 0.4912
Epoch 4/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 183s 580ms/step - accuracy: 0.7296 - loss: 0.5452 - val_accuracy: 0.6783 - val_loss: 0.6052
Epoch 5/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 176s 556ms/step - accuracy: 0.7619 - loss: 0.4961 - val_accuracy: 0.7976 - val_loss: 0.4624
Epoch 6/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 169s 536ms/step - accuracy: 0.7929 - loss: 0.4510 - val_accuracy: 0.8134 - val_loss: 0.4132
Epoch 7/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 169s 536ms/step - accuracy: 0.8115 - loss: 0.4121 - val_accuracy: 0.8586 - val_loss: 0.3300
Epoch 8/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 169s 535ms/step - accuracy: 0.8266 - loss: 0.3789 - val_accuracy: 0.8690 - val_loss: 0.3066
Epoch 9/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 169s 534ms/step - accuracy: 0.8598 - loss: 0.3229 - val_accuracy: 0.8942 - val_loss: 0.2514
Epoch 10/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 172s 544ms/step - accuracy: 0.8709 - loss: 0.2967 - val_accuracy: 0.9008 - val_loss: 0.2324
79/79 ━━━━━━━━━━━━━━━━━━━━ 10s 123ms/step - accuracy: 0.8996 - loss: 0.2319
Test Loss: 0.23236742615699768
Test Accuracy: 0.9007801413536072
```

Training accuracy = 89.96%
Testing accuracy = 90.08%

After callbacks

```
Epoch 1/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 172s 544ms/step - accuracy: 0.8791 - loss: 0.2782 - val_accuracy: 0.9070 - val_loss: 0.2275 - learning_rate: 9.0000e-04
Epoch 2/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 181s 571ms/step - accuracy: 0.8905 - loss: 0.2561 - val_accuracy: 0.9148 - val_loss: 0.2044 - learning_rate: 9.0000e-04
Epoch 3/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 180s 569ms/step - accuracy: 0.9001 - loss: 0.2374 - val_accuracy: 0.9264 - val_loss: 0.1841 - learning_rate: 9.0000e-04
Epoch 4/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 180s 568ms/step - accuracy: 0.9013 - loss: 0.2317 - val_accuracy: 0.8704 - val_loss: 0.2910 - learning_rate: 9.0000e-04
Epoch 5/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 180s 571ms/step - accuracy: 0.9076 - loss: 0.2162 - val_accuracy: 0.9304 - val_loss: 0.1770 - learning_rate: 9.0000e-04
Epoch 6/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 182s 575ms/step - accuracy: 0.9101 - loss: 0.2107 - val_accuracy: 0.9294 - val_loss: 0.1716 - learning_rate: 8.1000e-04
Epoch 7/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 189s 598ms/step - accuracy: 0.9177 - loss: 0.2002 - val_accuracy: 0.9216 - val_loss: 0.1853 - learning_rate: 8.1000e-04
Epoch 8/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 185s 586ms/step - accuracy: 0.9236 - loss: 0.1838 - val_accuracy: 0.9250 - val_loss: 0.1850 - learning_rate: 8.1000e-04
Epoch 9/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 184s 581ms/step - accuracy: 0.9214 - loss: 0.1910 - val_accuracy: 0.9336 - val_loss: 0.1598 - learning_rate: 8.1000e-04
Epoch 10/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 182s 577ms/step - accuracy: 0.9252 - loss: 0.1770 - val_accuracy: 0.9400 - val_loss: 0.1442 - learning_rate: 8.1000e-04
Epoch 11/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 182s 574ms/step - accuracy: 0.9266 - loss: 0.1800 - val_accuracy: 0.9356 - val_loss: 0.1536 - learning_rate: 7.2900e-04
Epoch 12/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 188s 595ms/step - accuracy: 0.9321 - loss: 0.1674 - val_accuracy: 0.9414 - val_loss: 0.1450 - learning_rate: 7.2900e-04
Epoch 13/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 194s 613ms/step - accuracy: 0.9328 - loss: 0.1601 - val_accuracy: 0.9290 - val_loss: 0.1714 - learning_rate: 7.2900e-04
```

Training accuracy = 93.28%

Testing accuracy = 92.90%

With image augmentation size=(150,150); batch_size=64; epochs=10

Training Accuracy	Testing Accuracy	**Validation Accuracy**
91.41%	93.06%	95%

These results of this model seem **worse** then the one with a resolution of **200 by 200 px.**

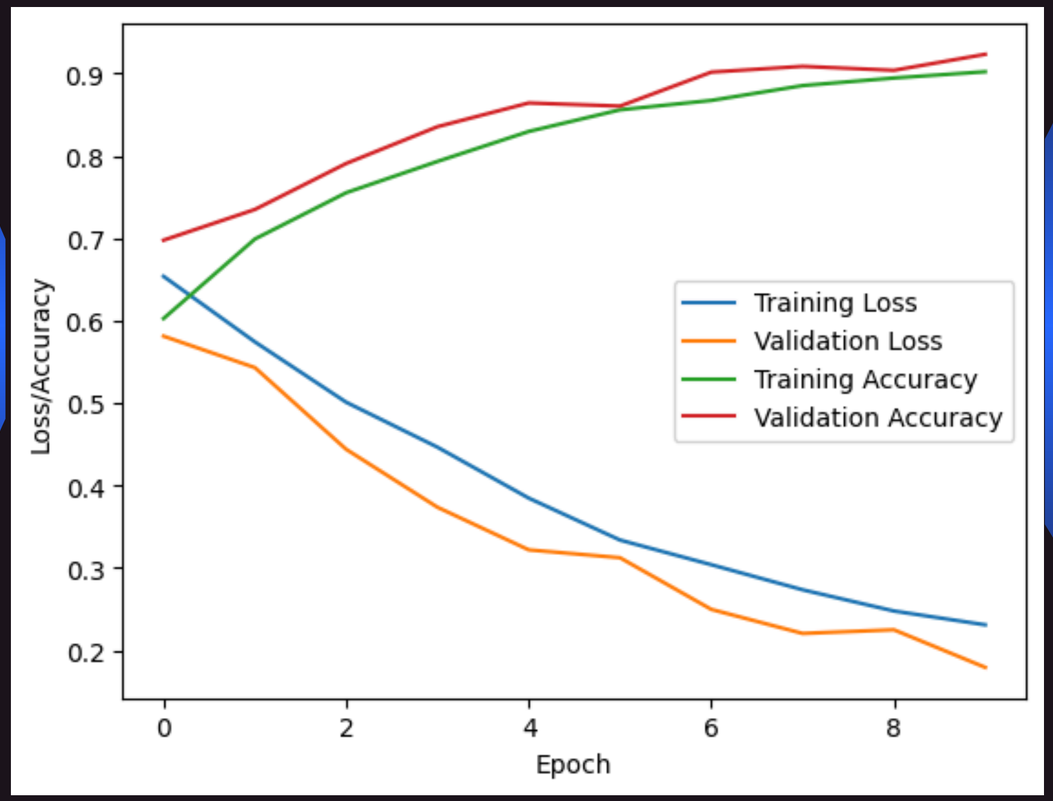
With the same structure, we increase the number of pixels to **256 by 256 px**

Layer (type)	Output Shape	Param #
conv2d_18 (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d_18 (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_19 (Conv2D)	(None, 125, 125, 64)	18,496
max_pooling2d_19 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_20 (Conv2D)	(None, 60, 60, 128)	73,856
max_pooling2d_20 (MaxPooling2D)	(None, 30, 30, 128)	0
conv2d_21 (Conv2D)	(None, 28, 28, 128)	147,584
max_pooling2d_21 (MaxPooling2D)	(None, 14, 14, 128)	0
conv2d_22 (Conv2D)	(None, 12, 12, 128)	147,584
max_pooling2d_22 (MaxPooling2D)	(None, 6, 6, 128)	0
flatten_4 (Flatten)	(None, 4608)	0
dense_8 (Dense)	(None, 512)	2,359,808
dense_9 (Dense)	(None, 2)	1,026

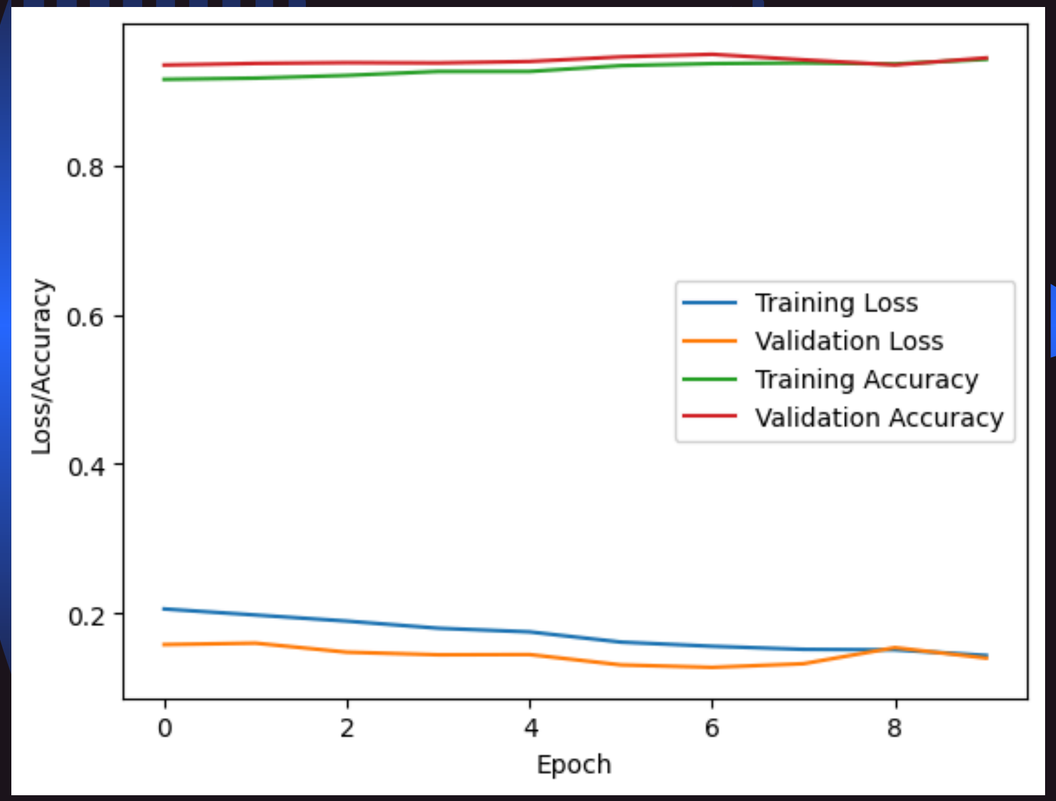
Total params: 8,247,752 (31.46 MB)

size=(256,256); batch_size=64; epochs=10 with augmentation

during training



during callbacks



Before callbacks

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 617s 2s/step - accuracy: 0.5690 - loss: 0.6769 - val_accuracy: 0.6977 - val_loss: 0.5813
Epoch 2/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 667s 2s/step - accuracy: 0.6720 - loss: 0.6013 - val_accuracy: 0.7349 - val_loss: 0.5431
Epoch 3/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 665s 2s/step - accuracy: 0.7482 - loss: 0.5124 - val_accuracy: 0.7908 - val_loss: 0.4440
Epoch 4/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 675s 2s/step - accuracy: 0.7855 - loss: 0.4593 - val_accuracy: 0.8354 - val_loss: 0.3737
Epoch 5/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 329s 1s/step - accuracy: 0.8220 - loss: 0.3934 - val_accuracy: 0.8642 - val_loss: 0.3221
Epoch 6/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 290s 919ms/step - accuracy: 0.8484 - loss: 0.3455 - val_accuracy: 0.8604 - val_loss: 0.3125
Epoch 7/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 282s 891ms/step - accuracy: 0.8588 - loss: 0.3194 - val_accuracy: 0.9016 - val_loss: 0.2498
Epoch 8/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 289s 915ms/step - accuracy: 0.8815 - loss: 0.2837 - val_accuracy: 0.9086 - val_loss: 0.2207
Epoch 9/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 286s 904ms/step - accuracy: 0.8927 - loss: 0.2567 - val_accuracy: 0.9038 - val_loss: 0.2251
Epoch 10/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 289s 915ms/step - accuracy: 0.8994 - loss: 0.2361 - val_accuracy: 0.9232 - val_loss: 0.1795
79/79 ━━━━━━━━━━━━━━━━━━━━ 14s 178ms/step - accuracy: 0.9159 - loss: 0.1945
Test Loss: 0.17949548363685608
Test Accuracy: 0.9231846332550049
```

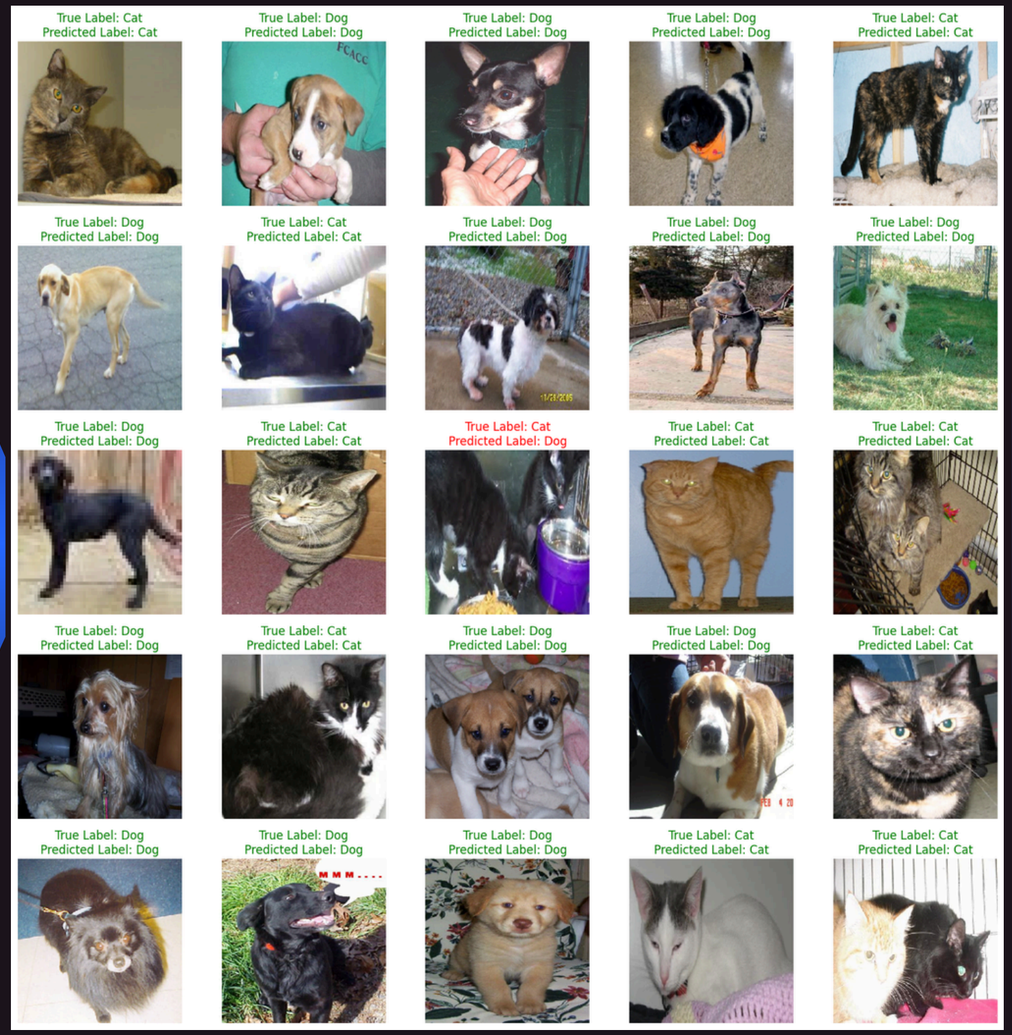
Training accuracy = 91.59%
Testing accuracy = 92.32%

After callbacks

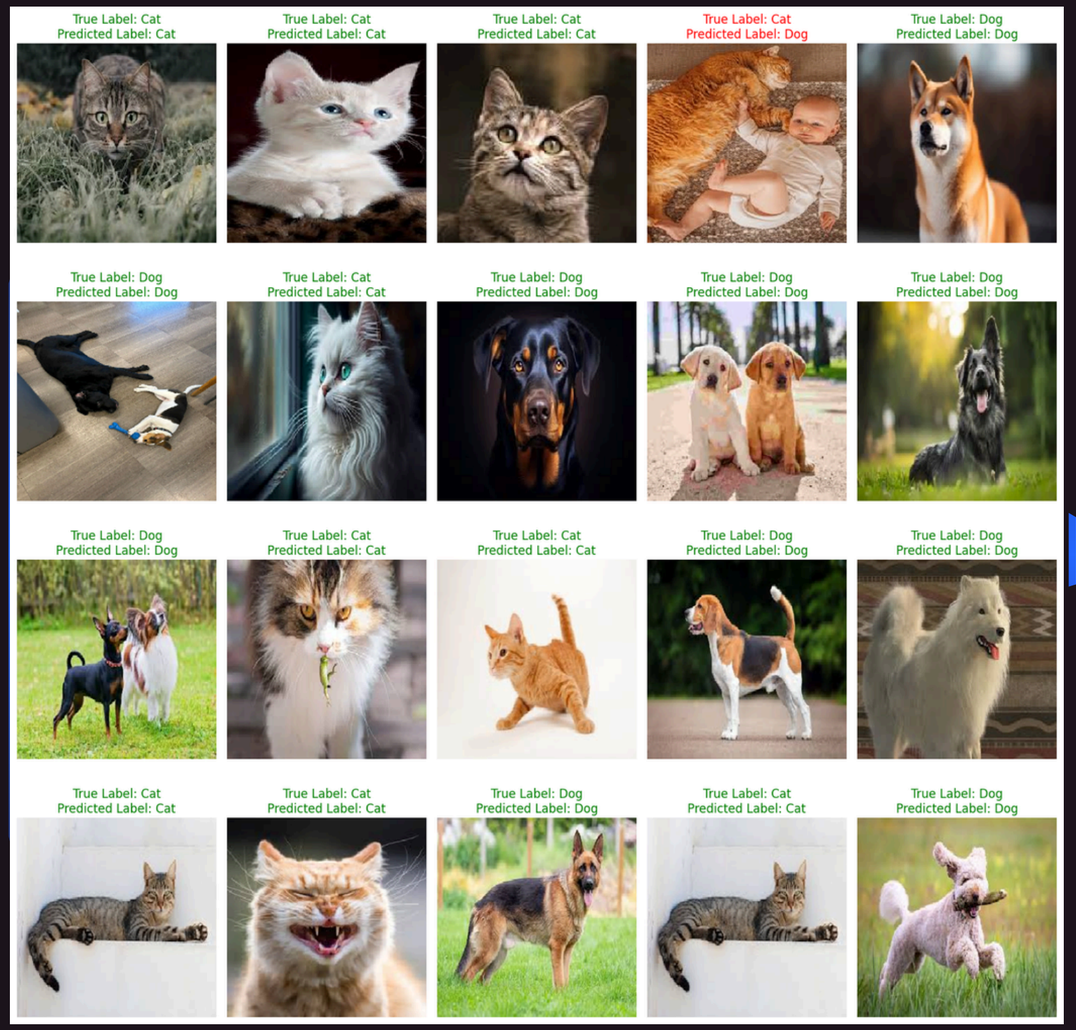
```
Epoch 1/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 294s 933ms/step - accuracy: 0.9157 - loss: 0.2067 - val_accuracy: 0.9340 - val_loss: 0.1584 - learning_rate: 9.0000e-04
Epoch 2/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 281s 890ms/step - accuracy: 0.9163 - loss: 0.1986 - val_accuracy: 0.9362 - val_loss: 0.1603 - learning_rate: 9.0000e-04
Epoch 3/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 285s 903ms/step - accuracy: 0.9189 - loss: 0.1917 - val_accuracy: 0.9372 - val_loss: 0.1482 - learning_rate: 9.0000e-04
Epoch 4/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 281s 889ms/step - accuracy: 0.9253 - loss: 0.1820 - val_accuracy: 0.9368 - val_loss: 0.1448 - learning_rate: 9.0000e-04
Epoch 5/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 300s 949ms/step - accuracy: 0.9270 - loss: 0.1748 - val_accuracy: 0.9388 - val_loss: 0.1451 - learning_rate: 9.0000e-04
Epoch 6/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 294s 932ms/step - accuracy: 0.9324 - loss: 0.1644 - val_accuracy: 0.9450 - val_loss: 0.1312 - learning_rate: 8.1000e-04
Epoch 7/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 299s 948ms/step - accuracy: 0.9353 - loss: 0.1556 - val_accuracy: 0.9484 - val_loss: 0.1279 - learning_rate: 8.1000e-04
Epoch 8/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 298s 942ms/step - accuracy: 0.9355 - loss: 0.1529 - val_accuracy: 0.9410 - val_loss: 0.1328 - learning_rate: 8.1000e-04
Epoch 9/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 289s 915ms/step - accuracy: 0.9371 - loss: 0.1501 - val_accuracy: 0.9342 - val_loss: 0.1543 - learning_rate: 8.1000e-04
Epoch 10/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 289s 915ms/step - accuracy: 0.9414 - loss: 0.1432 - val_accuracy: 0.9436 - val_loss: 0.1403 - learning_rate: 8.1000e-04
```

Training accuracy = 94.14%
Testing accuracy = 94.36%

size=(256,256); batch_size=64; epochs=10 with augmentation



Testing Accuracy= 93.75%



Validation Accuracy = 95%

We add a dropout layer of 50% with augmentation.

```python
# Model architecture
model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 3)),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(256, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(2, activation='softmax')
])
```

Training accuracy = 93.06%
Testing accuracy = 93.36%
Validation accuracy = 90%

Same structure with 20% dropout with augmentation

```python
# Model architecture
model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 3)),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(256, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.2),
        Dense(2, activation='softmax')
])
```

Training accuracy = 93.86%
Testing accuracy = 95%
Validation accuracy = 95%

We delete 1 ConvLayer and 1 MaxPooling Layer
We also double the number of filters in the last ConvLayer without any dropout layer

```python
# Model architecture
model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 3)),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(256, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(512, activation='relu'),
        Dense(2, activation='softmax')
])
```

Best Structure

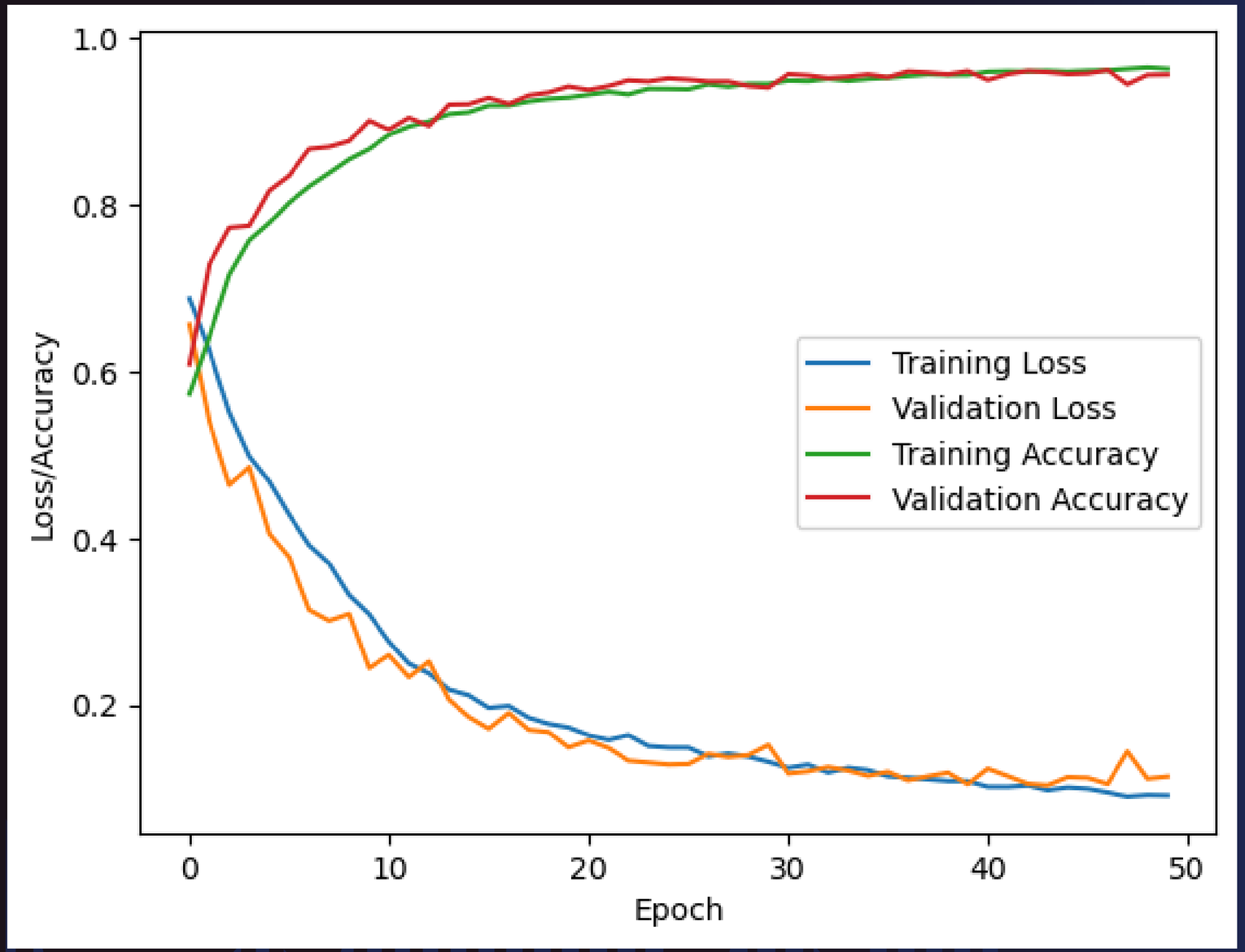
Layer (type)	Output Shape	Param #
conv2d_29 (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d_29 (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_30 (Conv2D)	(None, 125, 125, 64)	18,496
max_pooling2d_30 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_31 (Conv2D)	(None, 60, 60, 128)	73,856
max_pooling2d_31 (MaxPooling2D)	(None, 30, 30, 128)	0
conv2d_32 (Conv2D)	(None, 28, 28, 256)	295,168
max_pooling2d_32 (MaxPooling2D)	(None, 14, 14, 256)	0
flatten_6 (Flatten)	(None, 50176)	0
dense_12 (Dense)	(None, 512)	25,690,624
dense_13 (Dense)	(None, 2)	1,026

Total params: 78,240,200 (298.46 MB)

Test 1: after 30 epochs
Training accuracy = 94.56%
Testing accuracy = 94.96%
Validation accuracy = 95%

Test 2: after 50 epochs
Training accuracy = 96.29%
Testing accuracy = 95.60%%
Validation accuracy = 100.00%

```
Epoch 36/50
313/313 [==============================] - 179s 572ms/step - loss: 0.1145 - accuracy: 0.9522 - val_loss: 0.1198 - val_accuracy: 0.9528
Epoch 37/50
313/313 [==============================] - 180s 573ms/step - loss: 0.1128 - accuracy: 0.9543 - val_loss: 0.1096 - val_accuracy: 0.9598
Epoch 38/50
313/313 [==============================] - 178s 569ms/step - loss: 0.1109 - accuracy: 0.9563 - val_loss: 0.1145 - val_accuracy: 0.9584
Epoch 39/50
313/313 [==============================] - 179s 572ms/step - loss: 0.1088 - accuracy: 0.9555 - val_loss: 0.1189 - val_accuracy: 0.9562
Epoch 40/50
313/313 [==============================] - 180s 575ms/step - loss: 0.1087 - accuracy: 0.9556 - val_loss: 0.1050 - val_accuracy: 0.9602
Epoch 41/50
313/313 [==============================] - 180s 575ms/step - loss: 0.1019 - accuracy: 0.9594 - val_loss: 0.1236 - val_accuracy: 0.9496
Epoch 42/50
313/313 [==============================] - 179s 572ms/step - loss: 0.1018 - accuracy: 0.9600 - val_loss: 0.1147 - val_accuracy: 0.9566
Epoch 43/50
313/313 [==============================] - 179s 573ms/step - loss: 0.1036 - accuracy: 0.9596 - val_loss: 0.1054 - val_accuracy: 0.9604
Epoch 44/50
313/313 [==============================] - 183s 584ms/step - loss: 0.0978 - accuracy: 0.9605 - val_loss: 0.1036 - val_accuracy: 0.9590
Epoch 45/50
313/313 [==============================] - 182s 580ms/step - loss: 0.1012 - accuracy: 0.9596 - val_loss: 0.1134 - val_accuracy: 0.9566
Epoch 46/50
313/313 [==============================] - 184s 588ms/step - loss: 0.0996 - accuracy: 0.9606 - val_loss: 0.1126 - val_accuracy: 0.9574
Epoch 47/50
313/313 [==============================] - 183s 585ms/step - loss: 0.0951 - accuracy: 0.9611 - val_loss: 0.1052 - val_accuracy: 0.9614
Epoch 48/50
313/313 [==============================] - 179s 570ms/step - loss: 0.0902 - accuracy: 0.9628 - val_loss: 0.1443 - val_accuracy: 0.9444
Epoch 49/50
313/313 [==============================] - 186s 593ms/step - loss: 0.0921 - accuracy: 0.9645 - val_loss: 0.1115 - val_accuracy: 0.9556
Epoch 50/50
313/313 [==============================] - 179s 573ms/step - loss: 0.0916 - accuracy: 0.9629 - val_loss: 0.1141 - val_accuracy: 0.9560
```

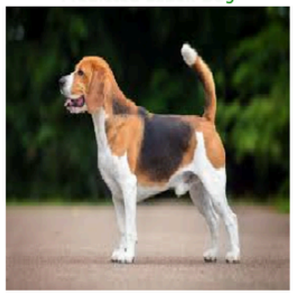

1.0
0.8
0.6
0.4
0.2
Loss/Accuracy
0
10
20
30
40
50
Epoch
Training Loss
Validation Loss
Training Accuracy
Validation Accuracy

True Label: Cat
Predicted Label: Cat

True Label: Dog
Predicted Label: Dog

True Label: Dog
Predicted Label: Dog

True Label: Dog
Predicted Label: Dog



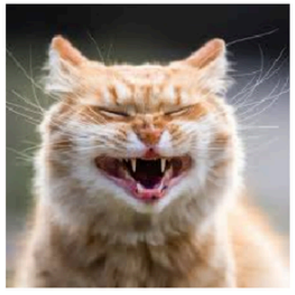
True Label: Dog
Predicted Label: Dog

True Label: Dog
Predicted Label: Dog

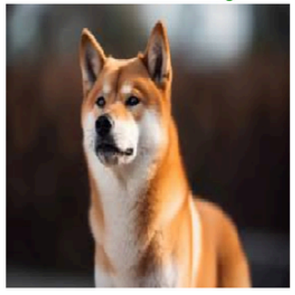
True Label: Cat
Predicted Label: Cat

True Label: Cat
Predicted Label: Cat

True Label: Cat
Predicted Label: Cat

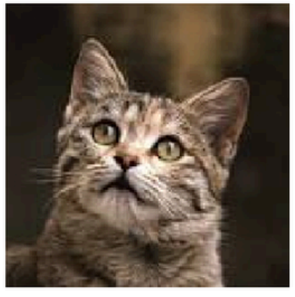


True Label: Dog
Predicted Label: Dog

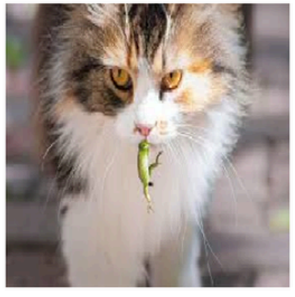


True Label: Cat
Predicted Label: Cat

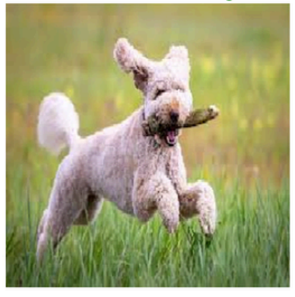
True Label: Cat
Predicted Label: Cat



True Label: Cat
Predicted Label: Cat



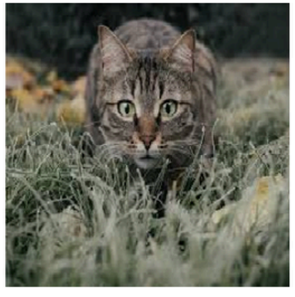
True Label: Dog
Predicted Label: Dog



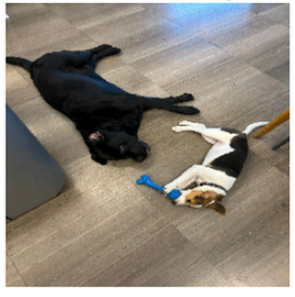
True Label: Cat
Predicted Label: Cat

True Label: Dog
Predicted Label: Dog

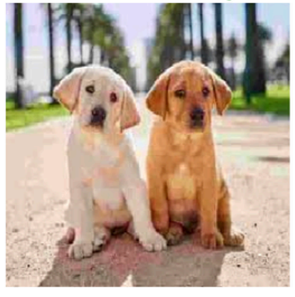
True Label: Cat
Predicted Label: Cat



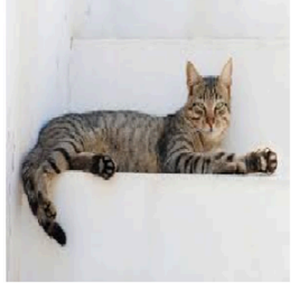
True Label: Dog
Predicted Label: Dog



True Label: Dog
Predicted Label: Dog



True Label: Cat
Predicted Label: Cat



Let’s test our code

Thank you for your attention!