Project Title: Cats vs Dogs Image Classification using CNN

🗒️ Objective

To build and train a Convolutional Neural Network (CNN) that can classify images as either cat or dog using a labeled dataset of pet images.

1. Data Preparation

a) Load preprocessed data stored in x.pickle and y.pickle

b) x contains grayscale image arrays of cats and dogs

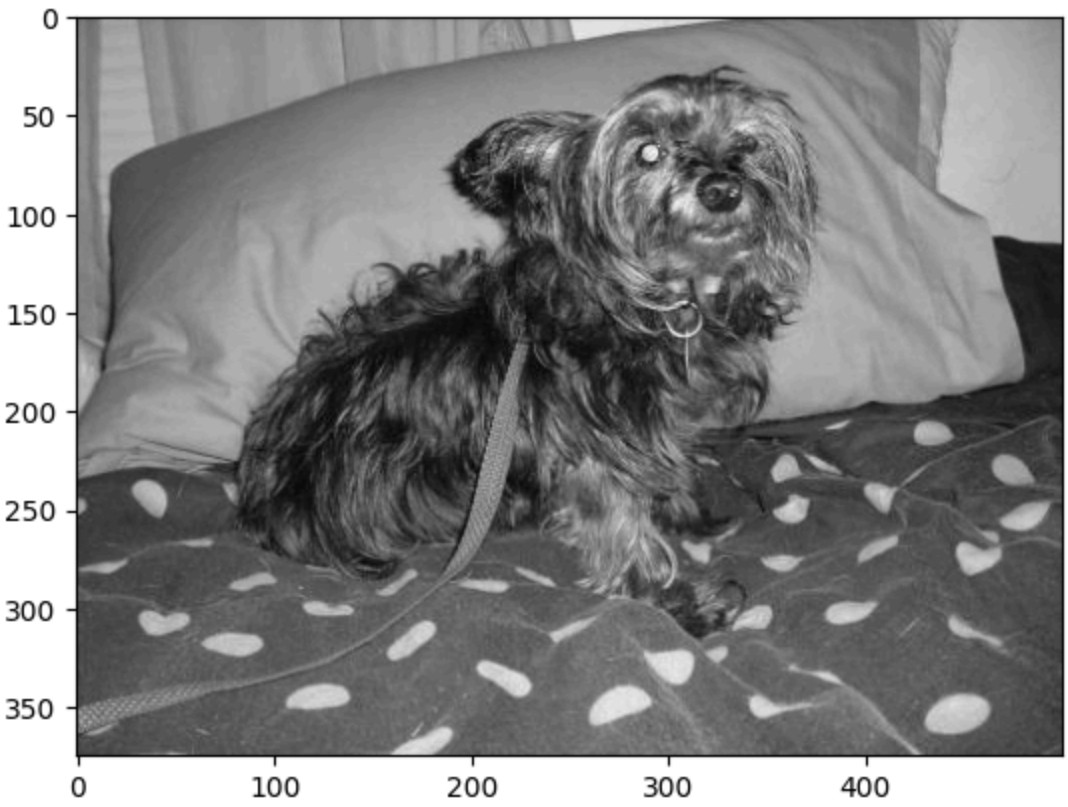
c) Normalize pixel values by dividing by 255.0

Note: Normalizing helps with faster and more stable training.

```
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2

DATADIR = "E:/Sentdex/PetImages"
CATEGORIES = ["Dog", "Cat"]

for categories in CATEGORIES:
    path = os.path.join(DATADIR, categories)
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
        plt.imshow(img_array, cmap = "gray")
        plt.show()
        break
    break
```



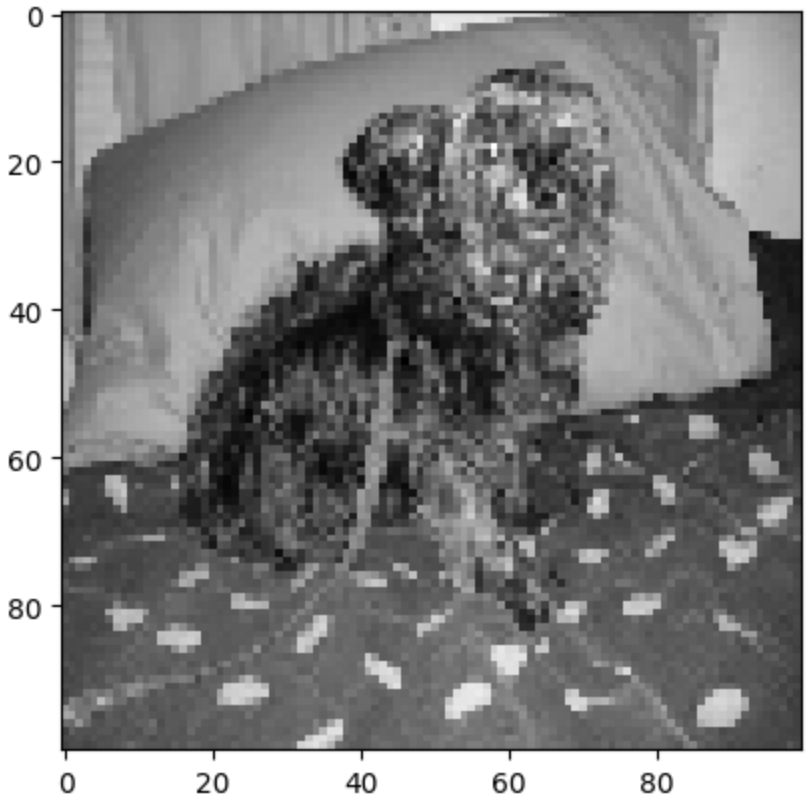
2. Have determined the shape of the image

```
print(img_array.shape)
```

```
(375, 500)
```

```
IMG_SIZE = 100

new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
plt.imshow(new_array, cmap = "gray")
plt.show()
```



3. Load and preprocess images with logging

```
training_data = []

def create_training_data():
    for categories in CATEGORIES:
        path = os.path.join(DATADIR, categories)
        class_num = CATEGORIES.index(categories)
        for img in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
                training_data.append([new_array, class_num])  # ✅ Correct spelling
            except Exception as e:
                pass

create_training_data()


print(len(training_data))
```

```
24946
```

4. Shuffle and split features/labels

```
import random

random.shuffle(training_data)


for sample in training_data[:10]:
    print(sample[1])
```

```
0
0
1
0
1
1
0
1
1
0
```

```
x = []
y = []


for features, label in training_data:
    x.append(features)
    y.append(label)
IMG_SIZE = 50
x = np.array(x).reshape(-1, IMG_SIZE, IMG_SIZE, 1)


import pickle

pickle_out = open("x.pickle","wb")
pickle.dump(x, pickle_out)
pickle_out.close()

pickle_out = open("y.pickle","wb")
pickle.dump(y, pickle_out)
pickle_out.close()


pickle_in = open("x.pickle","rb")
x = pickle.load(pickle_in)



pickle_in = open("y.pickle","rb")
y = pickle.load(pickle_in)
```

```
x[1]
```

```
array([[[116],
        [107],
        [118],
        ...,
        [112],
        [107],
        [ 95]],

       [[121],
        [110],
        [114],
        ...,
        [112],
        [107],
        [102]],

       [[103],
        [102],
        [119],
        ...,
        [113],
        [107],
        [104]],

       ...,

       [[ 60],
        [ 54],
        [ 79],
        ...,
        [ 89],
        [ 65],
        [ 60]],

       [[ 61],
        [ 61],
        [ 76],
        ...,
        [ 88],
        [ 61],
        [ 55]],

       [[ 54],
        [ 60],
        [ 64],
        ...,
        [ 96],
        [ 68],
        [ 56]]], dtype=uint8)
```

5. Building the CNN Model

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2

import pickle
import numpy as np

# Load your preprocessed data
x = pickle.load(open("x.pickle", "rb"))
y = pickle.load(open("y.pickle", "rb"))

# Normalize pixel values to range [0, 1]
x = x / 255.0

# Create a CNN model
model = Sequential()

model.add(Conv2D(64, (3, 3), input_shape=x.shape[1:]))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation("relu"))

model.add(Dense(1))
model.add(Activation("sigmoid"))

# Compile the model
#Loss Function: binary_crossentropy for 2 classes (cat/dog)
#Optimizer: adam for adaptive learning
#Metric: Accuracy to evaluate performance

model.compile(loss="binary_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])


# Convert to numpy arrays to avoid validation_split errors
x = np.array(x)
y = np.array(y)

# Train the model
```

```
model.fit(x, y, batch_size=32, validation_split=0.1, epochs=3)
#💉 Used 10% of data for validation to track model generalization.
```

```
Epoch 1/3
702/702 ━━━━━━━━━━━━━━━━━━━━ 198s 273ms/step - accuracy: 0.6045 - loss: 0.6499 - val_acc
Epoch 2/3
702/702 ━━━━━━━━━━━━━━━━━━━━ 192s 273ms/step - accuracy: 0.7686 - loss: 0.4895 - val_acc
Epoch 3/3
702/702 ━━━━━━━━━━━━━━━━━━━━ 259s 369ms/step - accuracy: 0.8043 - loss: 0.4255 - val_acc
<keras.src.callbacks.history.History at 0x24ee2254470>
```

3. Evaluation & Observations

I trained a Convolutional Neural Network (CNN) on a binary classification task using a dataset of cat and dog images. Over the course of 3 epochs, my model showed a clear improvement in both training and validation performance.

In the first epoch, the model started with a training accuracy of about 60.4% and a validation accuracy of 71.3%, which already showed it was beginning to learn useful features.

By the second epoch, training accuracy improved significantly to 76.8%, with the validation accuracy increasing to 76.9%, suggesting better generalization.

Finally, in the third epoch, training accuracy reached 80.4% and validation accuracy hit 78.6%, while the validation loss decreased to 0.4573, indicating the model was still improving without overfitting.

Hence, the model learned to distinguish between cats and dogs with reasonable accuracy after only a few epochs, and the gap between training and validation accuracy remained small — which is a good sign of generalization.

My Insights:

1. The loss steadily decreased, which shows that the model is minimizing its error.