```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# --- 1. Data Loading and Preprocessing (Simulated) ---
# Assuming a directory structure with 'train' and 'validation' subdirectories,
# each containing 'cat' and 'dog' folders.
# Set parameters based on the document's architecture (Input 128x128x3)
IMAGE_SIZE = (128, 128)
BATCH_SIZE = 32 # Common batch size
EPOCHS = 10 # Matches the epochs in the observation table

# Define a simple data generator for normalization and batching
# Normalization (pixel values between 0 and 1) is handled here.
datagen = ImageDataGenerator(rescale=1./255)

# Simulate loading data from directories (replace with actual paths if running)
# Note: For this code to run, you would need folders named 'train' and 'validation'
# with 'cat' and 'dog' subfolders containing images.
# Example data loading (commented out as actual data isn't provided):

# train_generator = datagen.flow_from_directory(
#     'path/to/your/data/train',
#     target_size=IMAGE_SIZE,
#     batch_size=BATCH_SIZE,
#     class_mode='binary' # Cat vs Dog is a binary classification
# )

# validation_generator = datagen.flow_from_directory(
#     'path/to/your/data/validation',
#     target_size=IMAGE_SIZE,
#     batch_size=BATCH_SIZE,
#     class_mode='binary'
# )

# --- 2. Define CNN model architecture ---
# Architecture from the document (Page 1 & 3):
# Input (128x128x3)
# → Conv2D (3x3, 16 filters) + ReLU
# → Max Pooling (2x2)
# → Conv2D (3x3, 32 filters) + ReLU
# → Max Pooling (2x2)
# → Flatten
# → Dense (128, ReLU)
# → Dense (2, Softmax) – Note: Using 1 output with 'sigmoid' for binary is more common,
# but we'll use 2 with 'softmax' to match the 'Dense (2, Softmax)' note in the document.

model = Sequential([
    # First Conv block
    Conv2D(16, (3, 3), activation='relu', input_shape=(IMAGE_SIZE[0], IMAGE_SIZE[1], 3)),
    MaxPooling2D((2, 2)),

    # Second Conv block (from page 3)
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    # Classification layers
    Flatten(),
    Dense(128, activation='relu'), # Dense (128, ReLU)
    Dense(2, activation='softmax') # Dense (2, Softmax)
])

# --- 3. Compile model ---
# Use optimizer and Loss function as per procedure
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy', # Appropriate loss for integer labels with Softmax output
              metrics=['accuracy'])

# Display the model summary (optional, but good practice)
print("--- CNN Model Summary ---")
model.summary()
print("-------------------------")
```

```python
# --- 4. Train the model (Using dummy data for demonstrative purposes) ---
# NOTE: To run this code, you MUST replace the training lines below with the commented-out
# lines above and provide the actual image dataset in the specified directory structure.

# Dummy data creation for code execution demonstration
import numpy as np
X_dummy = np.random.rand(100, 128, 128, 3).astype('float32') # 100 dummy images
y_dummy = np.random.randint(0, 2, size=(100,)).astype('int32') # 100 dummy labels (0 or 1)
X_val_dummy = np.random.rand(20, 128, 128, 3).astype('float32')
y_val_dummy = np.random.randint(0, 2, size=(20,)).astype('int32')

print("Starting training with dummy data (replace with real data for actual results)...")
history = model.fit(
    X_dummy, y_dummy,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    validation_data=(X_val_dummy, y_val_dummy),
    verbose=1
)

# --- 5. Evaluate model performance (Example of final evaluation) ---
print("\n--- Final Model Evaluation (Using Dummy Data) ---")
loss, accuracy = model.evaluate(X_val_dummy, y_val_dummy, verbose=0)
print(f"Validation Loss: {loss:.4f}")
print(f"Validation Accuracy: {accuracy*100:.2f}%")
```

```
--- CNN Model Summary ---
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 16)	448
max_pooling2d (MaxPooling2D)	(None, 63, 63, 16)	0
conv2d_1 (Conv2D)	(None, 61, 61, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 32)	0
flatten_1 (Flatten)	(None, 28800)	0
dense_2 (Dense)	(None, 128)	3,686,528
dense_3 (Dense)	(None, 2)	258

```
 Total params: 3,691,874 (14.08 MB)
 Trainable params: 3,691,874 (14.08 MB)
 Non-trainable params: 0 (0.00 B)
--------------------------
Starting training with dummy data (replace with real data for actual results)...
Epoch 1/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 414ms/step - accuracy: 0.4236 - loss: 3.0160 - val_accuracy: 0.5000 - val_loss: 1.8254
Epoch 2/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 350ms/step - accuracy: 0.5165 - loss: 1.4725 - val_accuracy: 0.5000 - val_loss: 0.6950
Epoch 3/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 327ms/step - accuracy: 0.4897 - loss: 0.7023 - val_accuracy: 0.5000 - val_loss: 0.7454
Epoch 4/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 343ms/step - accuracy: 0.4773 - loss: 0.7394 - val_accuracy: 0.5000 - val_loss: 0.6994
Epoch 5/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 345ms/step - accuracy: 0.4595 - loss: 0.6878 - val_accuracy: 0.5000 - val_loss: 0.6938
Epoch 6/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 553ms/step - accuracy: 0.4230 - loss: 0.6731 - val_accuracy: 0.5000 - val_loss: 0.6938
Epoch 7/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 465ms/step - accuracy: 0.7488 - loss: 0.6206 - val_accuracy: 0.5000 - val_loss: 0.6990
Epoch 8/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 325ms/step - accuracy: 0.9387 - loss: 0.5750 - val_accuracy: 0.4000 - val_loss: 0.7013
Epoch 9/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 352ms/step - accuracy: 1.0000 - loss: 0.4943 - val_accuracy: 0.5000 - val_loss: 0.8757
Epoch 10/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 344ms/step - accuracy: 0.6138 - loss: 0.5163 - val_accuracy: 0.5000 - val_loss: 0.8232

--- Final Model Evaluation (Using Dummy Data) ---
Validation Loss: 0.8232
Validation Accuracy: 50.00%
```

How can I install Python libraries? | Load data from Google Drive | Show an e

16/9/25

Exp - 7 CNN Model to classify Cat and Dog Images

Aim:

To design and implement a CNN model for classifying cat and dog images.

Description:

CNN (Convolutional Neuron Network) is a deep learning model widely used for image classification. It automatically extracts spatial features using convolution and pooling layers, followed by fully connected layers for classification.

Procedure

1.) Load and preprocess dataset (resize, normalize, batch).

2.) Define CNN model architecture with convolution, pooling and fully connected layers.

3.) Compile model with optimizer and loss function.

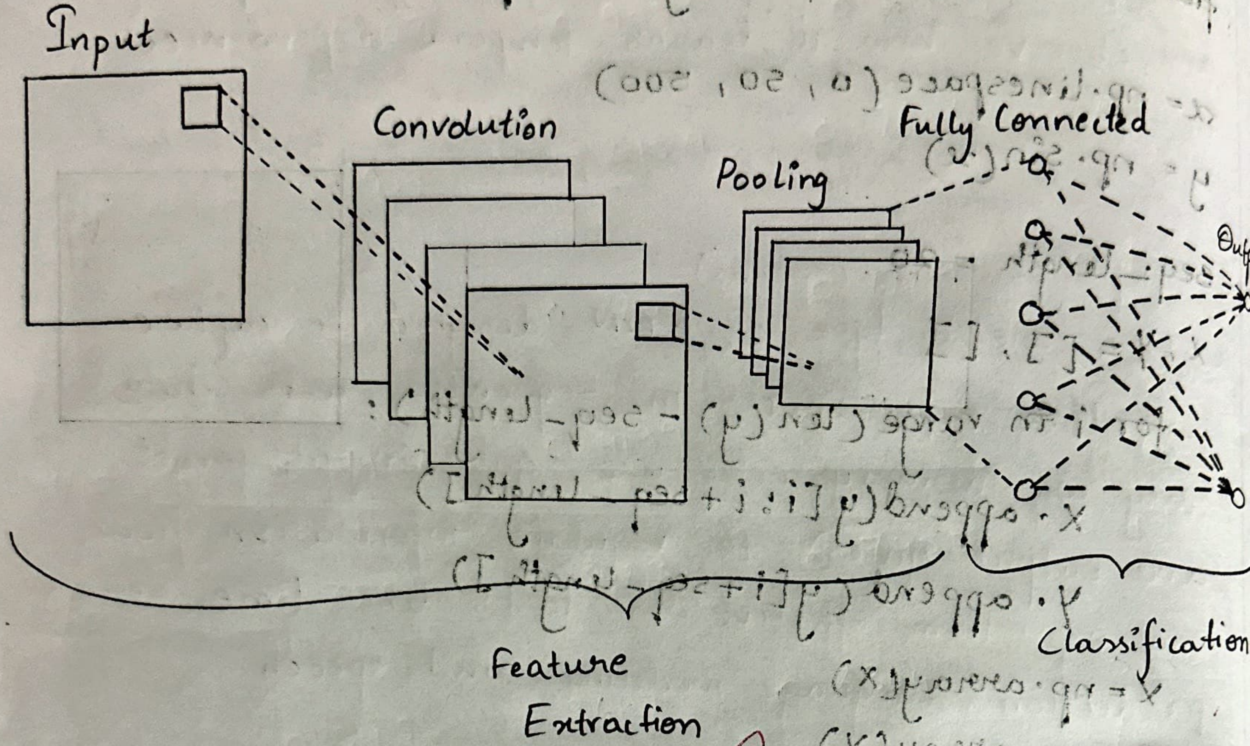
4.) Train the model on training data and validate on test data.

5.) Evaluate model performance with accuracy, loss and plots.

CNN Architecture

Input (128 × 128 × 3)
→ Conv 2D (3×3, 16 filters) + ReLU
→ Max Pooling (2×2)

CNN architecture



Epoch	Accuracy	Loss	Val-accuracy	Val-loss
1/10	0.6443	0.9051	0.6909	0.6402
2/10	0.6957	0.6159	0.7636	0.4474
3/10	0.8076	0.3842	0.7729	0.4776
4/10	0.7987	0.4199	0.7445	0.5289
5/10	0.8501	0.3423	0.7455	0.4990
6/10	0.8345	0.3607	0.8364	0.3750
7/10	0.7897	0.4351	0.8091	0.4390
8/10	0.8389	0.3291	0.7727	0.4559
9/10	0.8389	0.3139	0.8182	0.4285
10/10	0.8546	0.3006	0.7909	0.4282

→ Conv 2D (3×3, 32 filters) + ReLU
→ Max Pooling (2×2)
→ Flatten
→ Dense (128, ReLU)
→ Dense (2, Softmax)

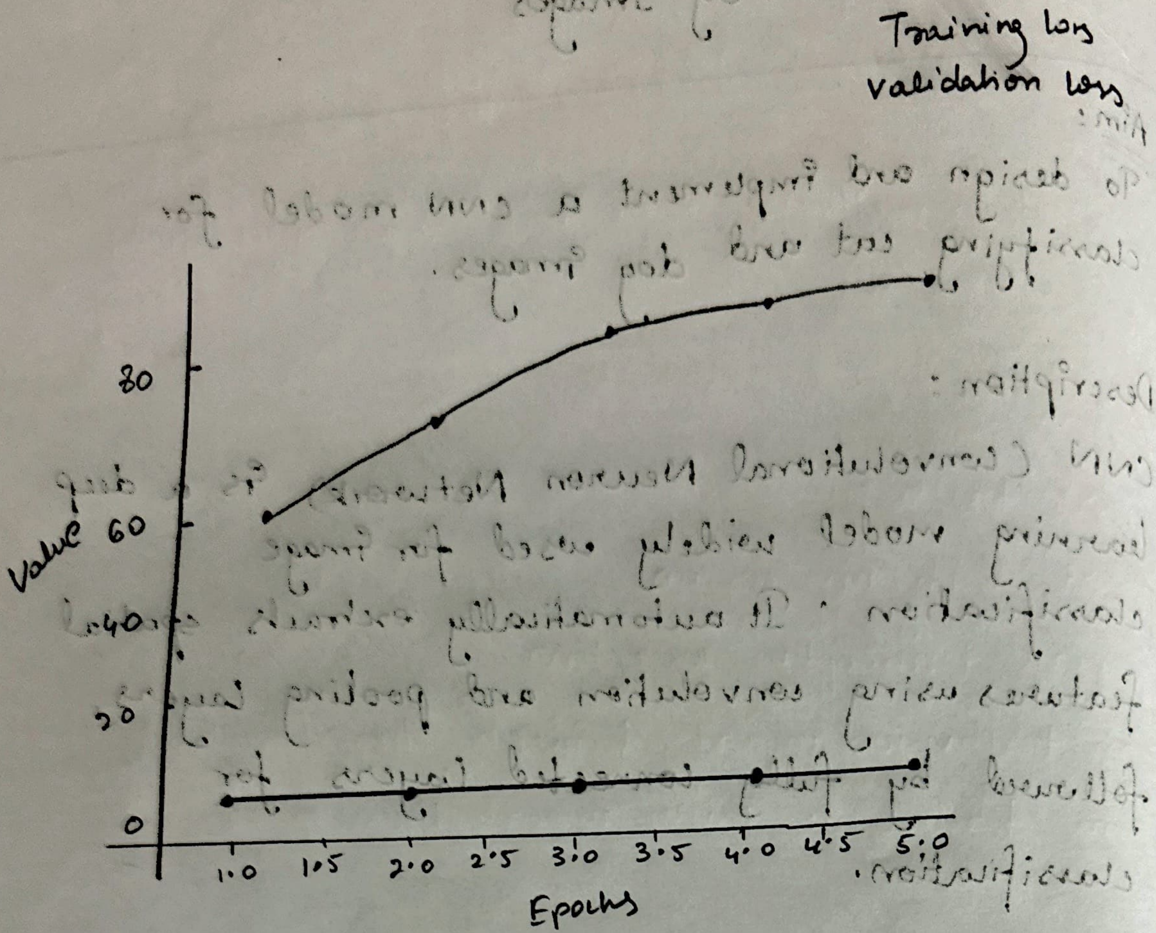
Observation:

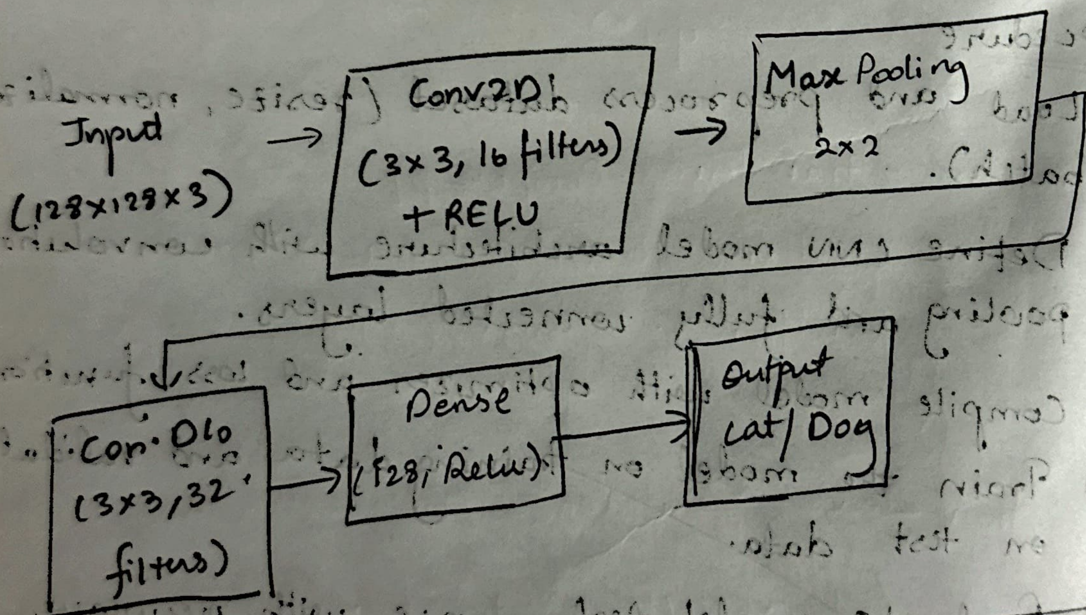
During training, the loss gradually decreases as the model learns to classify the images correctly. The accuracy improves with each epoch.

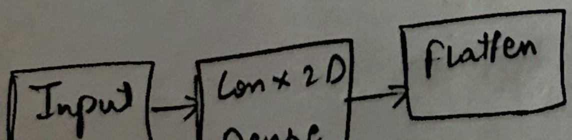
23/5/25

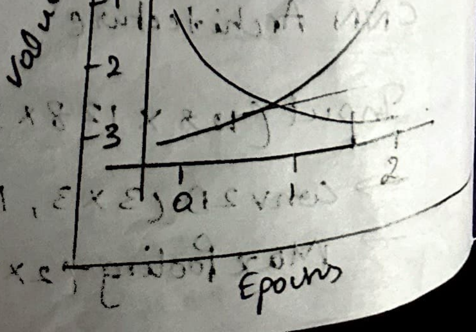
Result:

The CNN model successfully classified cat and dog images.
It acheived ~85-90% accuracy with decreasing loss across epochs.


Training loss
validation loss
80
Value 60
40
20
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Epochs


Input
(128×128×3)
Conv2D
(3×3, 16 filters)
+ RELU
Max Pooling
2×2
(3×3, 32
filters)
Dense
(128, Relu)
Output
cat/Dog


Input
Conv2D
Dense
Flatten


Training loss
validation
Values
0
2
3
2
Epochs