EXPERIMENT-7 23-9-25

Aim: To build a convolutional neural network model that can classify images of cats and dogs using a labelled dataset.

Description:

A CNN is a deep learning algorithm designed to recognize patterns in visual data. ~~CNN are inspired~~ The core idea is to use a series of convolutions over input images to capture features such as edges, textures, and higher-level structures.

architecture -

- Convolution layer - apply filters to images to extract features.
- Activation (ReLU layer) - Adds non-linearity to the network after each convolution.
- Pooling layer - Reduces the spatial dimension of the input, helping reduce computational complexity and overfitting.
- F.C layer - After feature extraction, flattened o/p is passed through FC layer to predict labeled class.
- Softmax func. - Used ~~to~~ in output layer to convert the final layers values into probabilities aiding in classification.

Pseudocode:

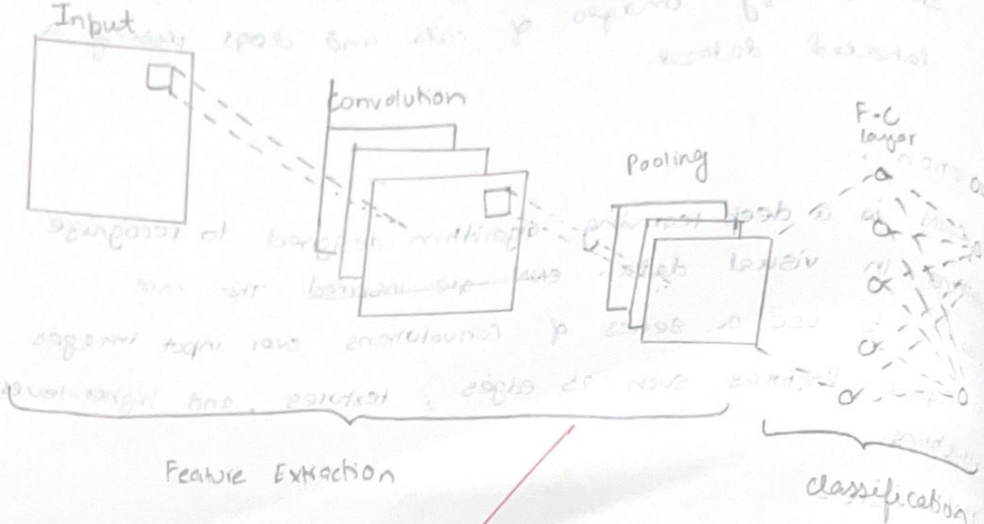
1. Define CNN architecture

   create a neural network with convolution, ReLU, pooling layers. Flatten the output and add two fully connected layers.

2. Preprocess the data

   - Resize all images
   - Convert images to tensors
   - normalization
   - create data loaders.

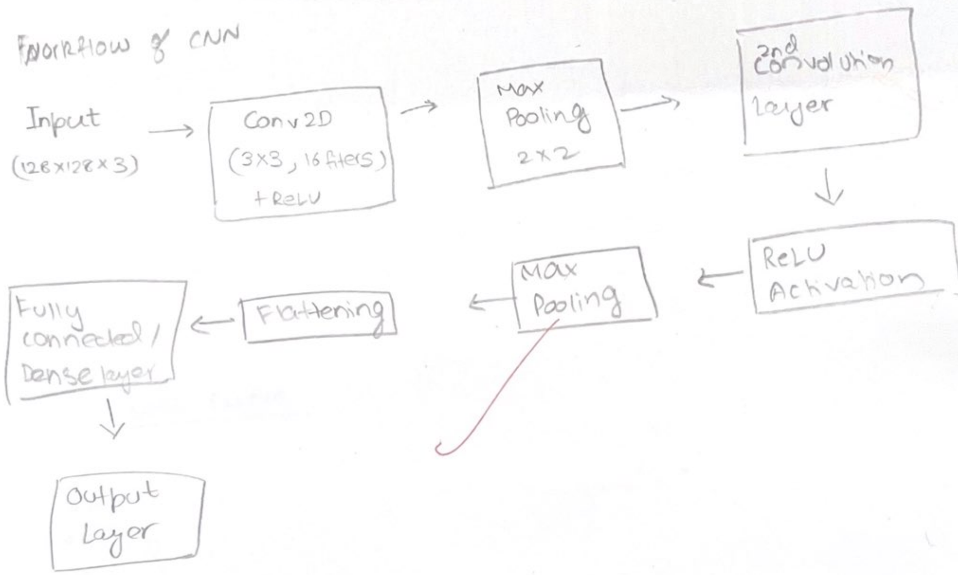
CNN Architecture

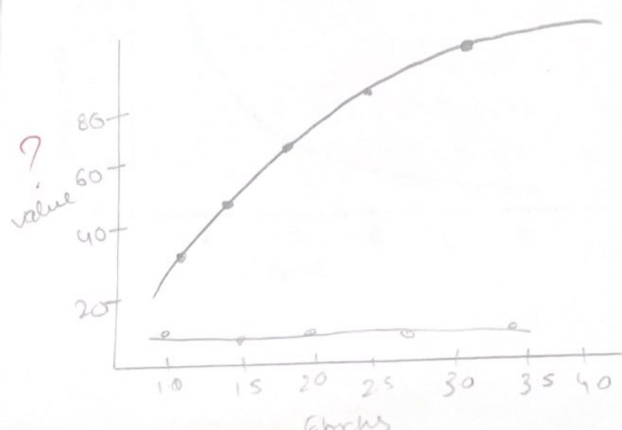


Output:

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	0.4149	0.7445	0.5289
5/10	0.8501	0.3423	0.7455	0.4910
6/10	0.8845	0.3607	0.8364	0.3750
7/10	0.7897	0.4351	0.8091	0.4396
8/10	0.8389	0.3291	0.7727	0.4359
9/10	0.8389	0.3139	0.8182	0.4285
10/10	0.8546	0.3006	0.7909	0.4182

Workflow of CNN





3. Set-up training
initialize model, define loss fnc., choose an optimizer

4. Train the model

5. Evaluate the model

Observation :

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

Result:

CNN model successfully classifies images of cats and dogs with a reasonable level of accuracy demonstrating the power of convolutional layers for image recognition tasks.

7/23/9

Arc File Edit View

Week7.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▷ Run all

```
# Install Kaggle API if not available
!pip install kaggle

# Make a kaggle.json file from your Kaggle account (Account → API → Create New API Token)
# Upload kaggle.json
from google.colab import files
files.upload()

# Move kaggle.json to correct path
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

Show hidden output

```
!kaggle datasets download -d mahdinavaei/cat-and-dog-classifier
!unzip cat-and-dog-classifier.zip -d /content/data/
```

Show hidden output

```
data = "/content/data/data"

import tensorflow as tf
import os
import matplotlib.pyplot as plt
import cv2
import imghdr

%matplotlib inline
```

```
/tmp/ipython-input-1101764521.py:7: DeprecationWarning: 'imghdr' is deprecated and slated for removal in Pyth
  import imghdr
```

Variables Terminal

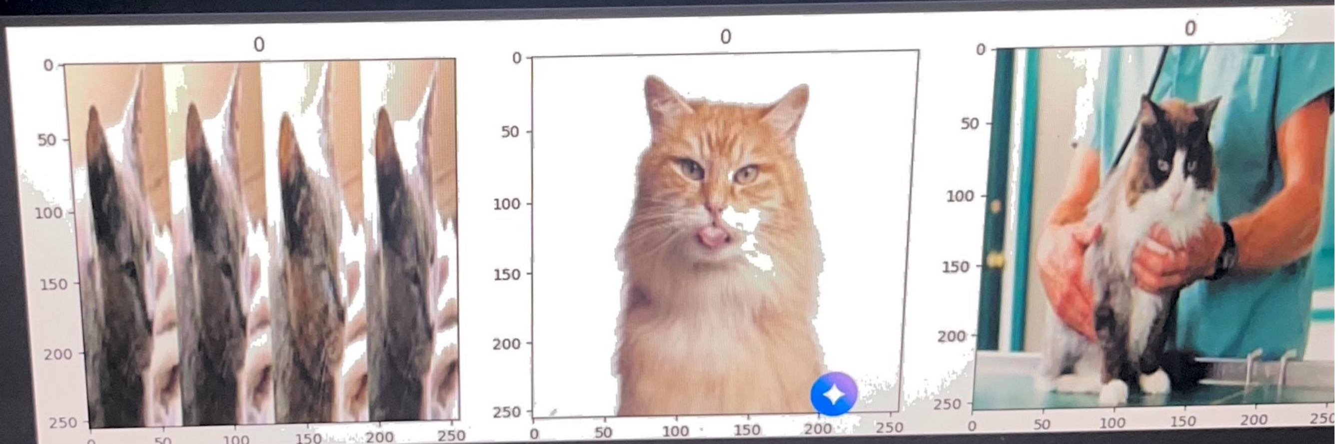
```
/tmp/ipython-input-1101764521.py:7: DeprecationWarning: 'imghdr' is deprecated and slated for removal in
  import imghdr
```

```python
# Load image data into a TensorFlow dataset
data = tf.keras.utils.image_dataset_from_directory('/content/data/data')

data_iterator = data.as_numpy_iterator()
batch = data_iterator.next()
```

```
Found 1152 files belonging to 2 classes.
```

```python
fig, ax = plt.subplots(ncols=4, figsize=(20,20))
for idx, img in enumerate(batch[0][:4]):
    ax[idx].imshow(img.astype(int))
    ax[idx].title.set_text(batch[1][idx])
```



{} Variables   Terminal

```
data = data.map(lambda x,y: (x/255, y))
data.as_numpy_iterator().next()
```

Show hidden output

```
train_size = int(len(data)*.7)
val_size = int(len(data)*.2)
test_size = int(len(data)*.1)
```

```
train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size+val_size).take(test_size)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
model = Sequential()
```

```
model.add(Conv2D(16, (3,3), 1, activation='relu', padding = 'same', input_shape=(256,256
model.add(MaxPooling2D(pool_size = (2,2), strides=(2,2)))
model.add(Conv2D(32, (3,3), 1, activation='relu', padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2), strides=(2,2)))
model.add(Conv2D(16, (3,3), 1, activation='relu', padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2), strides=(2,2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113:
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
model.compile('adam', loss=tf.losses.BinaryCrossentropy(), metrics=['accuracy'])
```

Variables   Terminal

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 256, 256, 16)	448
max_pooling2d (MaxPooling2D)	(None, 128, 128, 16)	0
conv2d_1 (Conv2D)	(None, 128, 128, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_2 (Conv2D)	(None, 64, 64, 16)	4,624
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 16)	0
flatten (Flatten)	(None, 16384)	0
dense (Dense)	(None, 256)	4,194,560
dense_1 (Dense)	(None, 1)	257

Total params: 4,204,529 (16.04 MB)
Trainable params: 4,204,529 (16.04 MB)
Non-trainable params: 0 (0.00 B)

[ ]

```
logdir='logs'
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
```

[ ]

```
hist = model.fit(train, epochs=50, validation_data=val, callbacks=[tensorboard_callback])
print(hist.history) # this will print a dictionary object, now you need to grab the metrics / score y

# if your score == 'acc', if not replace 'acc' with your metric

best_score = max(hist.history['val_accuracy'])

print(f"Best Validation score is: {best_score}")
```

{ } Variables   Terminal

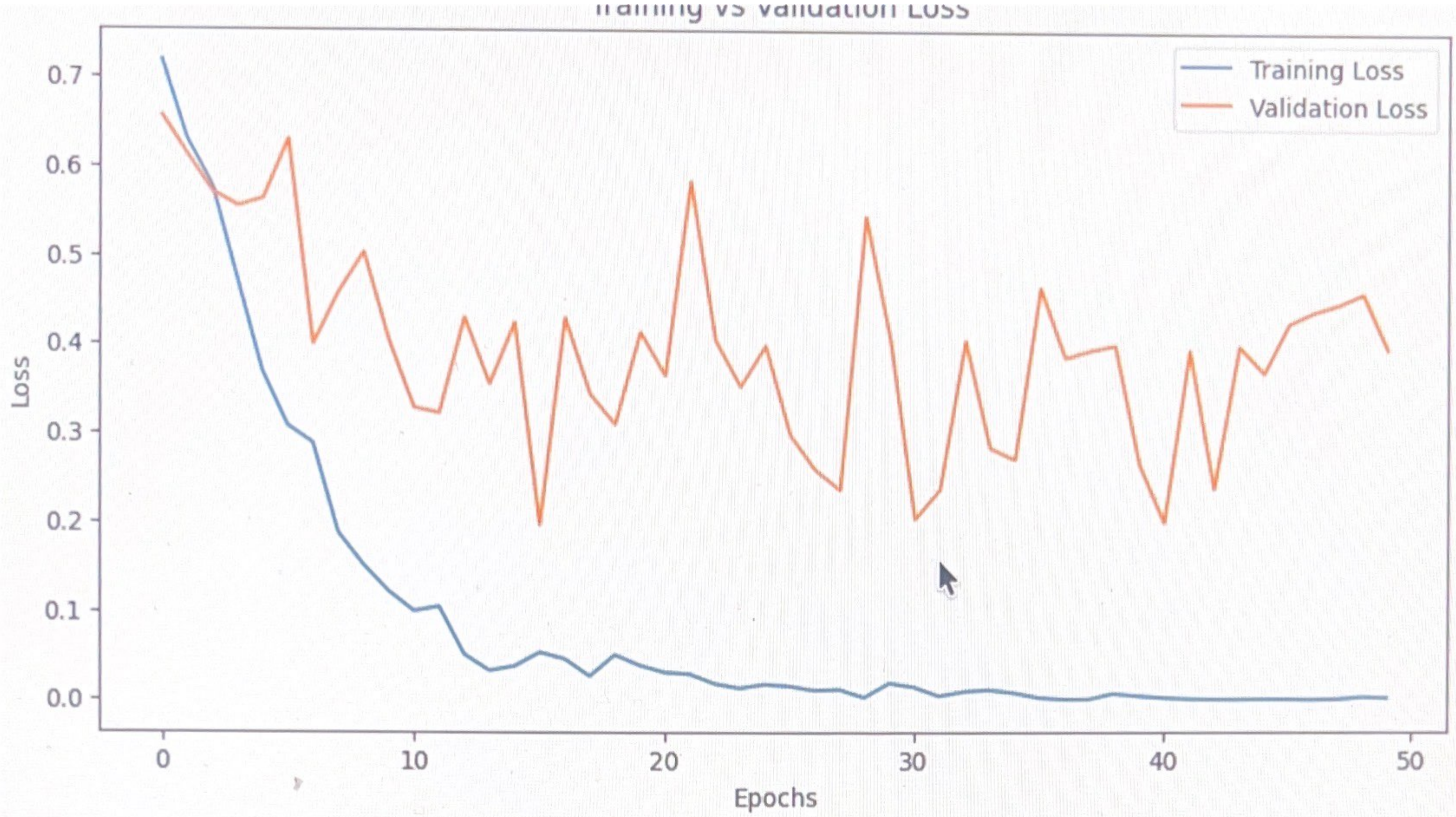
```
Epoch 1/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 8s 340ms/step - accuracy: 0.9999 - loss: 8.4385e-04 - val_accura
Epoch 2/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 7s 306ms/step - accuracy: 0.9984 - loss: 0.0036 - val_accuracy:
Epoch 3/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 8s 337ms/step - accuracy: 0.9991 - loss: 0.0032 - val_accuracy: 0
Epoch 4/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 7s 288ms/step - accuracy: 0.9973 - loss: 0.0192 - val_accuracy: 0
Epoch 5/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 8s 343ms/step - accuracy: 0.9853 - loss: 0.0417 - val_accuracy: 0
Epoch 6/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 7s 290ms/step - accuracy: 0.9818 - loss: 0.0603 - val_accuracy: 0.
Epoch 7/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 11s 420ms/step - accuracy: 0.9750 - loss: 0.0780 - val_accuracy: 0
Epoch 8/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 8s 340ms/step - accuracy: 0.9801 - loss: 0.0612 - val_accuracy: 0.9
Epoch 9/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 7s 272ms/step - accuracy: 0.9943 - loss: 0.0204 - val_accuracy: 0.9
Epoch 10/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 8s 324ms/step - accuracy: 0.9960 - loss: 0.0111 - val_accuracy: 0.9
Epoch 11/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 7s 277ms/step - accuracy: 0.9984 - loss: 0.0086 - val_accuracy: 0.92
Epoch 12/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 9s 375ms/step - accuracy: 0.9979 - loss: 0.0140 - val_accuracy: 0.91
Epoch 13/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 8s 340ms/step - accuracy: 0.9960 - loss: 0.0248 - val_accuracy: 0.919
Epoch 14/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 8s 303ms/step - accuracy: 0.9903 - loss: 0.0206 - val_accuracy: 0.897
Epoch 15/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 8s 344ms/step - accuracy: 0.9987 - loss: 0.0101 - val_accuracy: 0.933
Epoch 16/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 7s 288ms/step - accuracy: 0.9906 - loss: [illegible]311 - val_accuracy: 0.9062
Epoch 17/50
```

es Terminal

```python
import matplotlib.pyplot as plt

# Plot Loss
plt.figure(figsize=(10,5))
plt.plot(hist.history['loss'], label='Training Loss')
plt.plot(hist.history['val_loss'], label='Validation Loss')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training vs Validation Loss")
plt.legend()
plt.show()

# Plot Accuracy
plt.figure(figsize=(10,5))
plt.plot(hist.history['accuracy'], label='Training Accuracy')
plt.plot(hist.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training vs Validation Accuracy")
plt.legend()
plt.show()
```



Training vs Validation Accuracy