Experiment 7 Build a CNN Model
16/9/25 for Cat & Dog Classification.

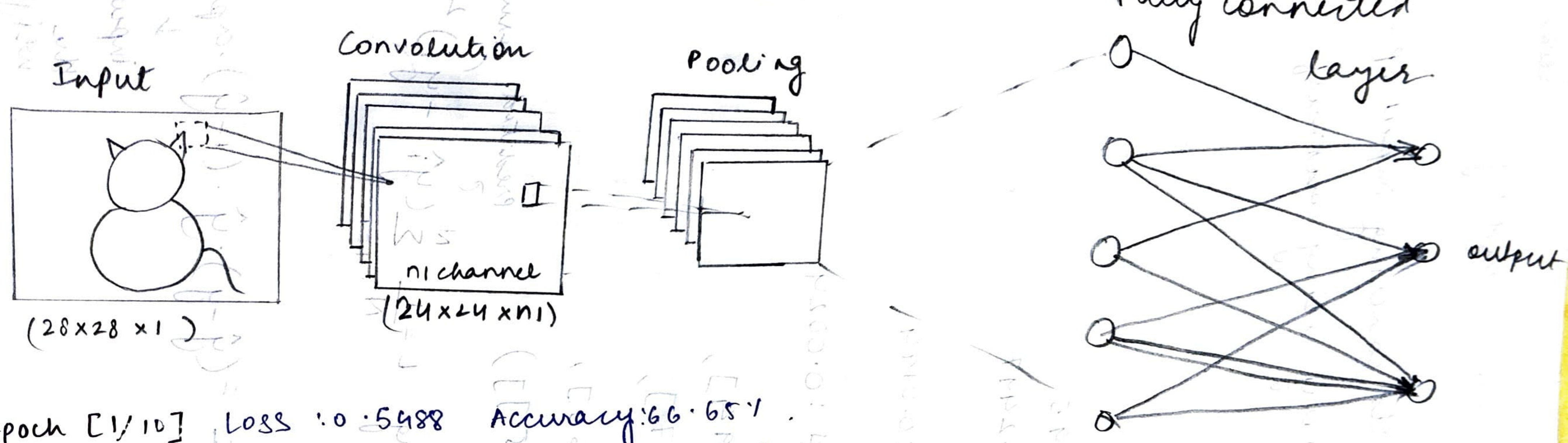
Aim: To design, develop and evaluate a CNN model capable of accurately classifying images of cats and dogs.

Objective
→ Dataset preparation
→ Model Development
→ Model Training and optimization
→ Model Evaluation

Pseudocode
→ Importing tensorflow-datasets as tfds
→ Set data_dir_base to the directory where your Cats & Dogs dataset is stored
→ Import pytorch and necessary libraries
→ Resize all images to 128 X 128
→ Convert images to tensors
→ Normalize pixel values
→ Load cats-vs-dogs dataset
→ 80% training 20% validation split → Validation 4652
↓ Train samples 18610

→ Convert Tensorflow Tensors to pytorch Tensors
→ Define CNN with parameters
↳ conv. layer
↳ max pooling
↳ conv. layer
↳ conv. layer
↳ dropout layer with 0.5 probability
→ Calculate flattened feature size
→ Created fully connected layer.
→ Apply ReLU do forward pass

CNN Architecture.



Epoch [1/10] Loss : 0.5488 Accuracy: 66.65%.

epoch [2/10] Avg Loss : 0.4664 Accuracy : 76.30%.

epoch [3/10] Avg Loss : 0.4639 Accuracy : 77.8%.

epoch [4/10] Avg Loss : 0.3149 Accuray 85.28%.

epoch [5/10] Avg Loss : 0.2454 Accuray : 89.20%.

epoch [6/10] Avg Loss : 0.1801 Accuray : 92.25%.

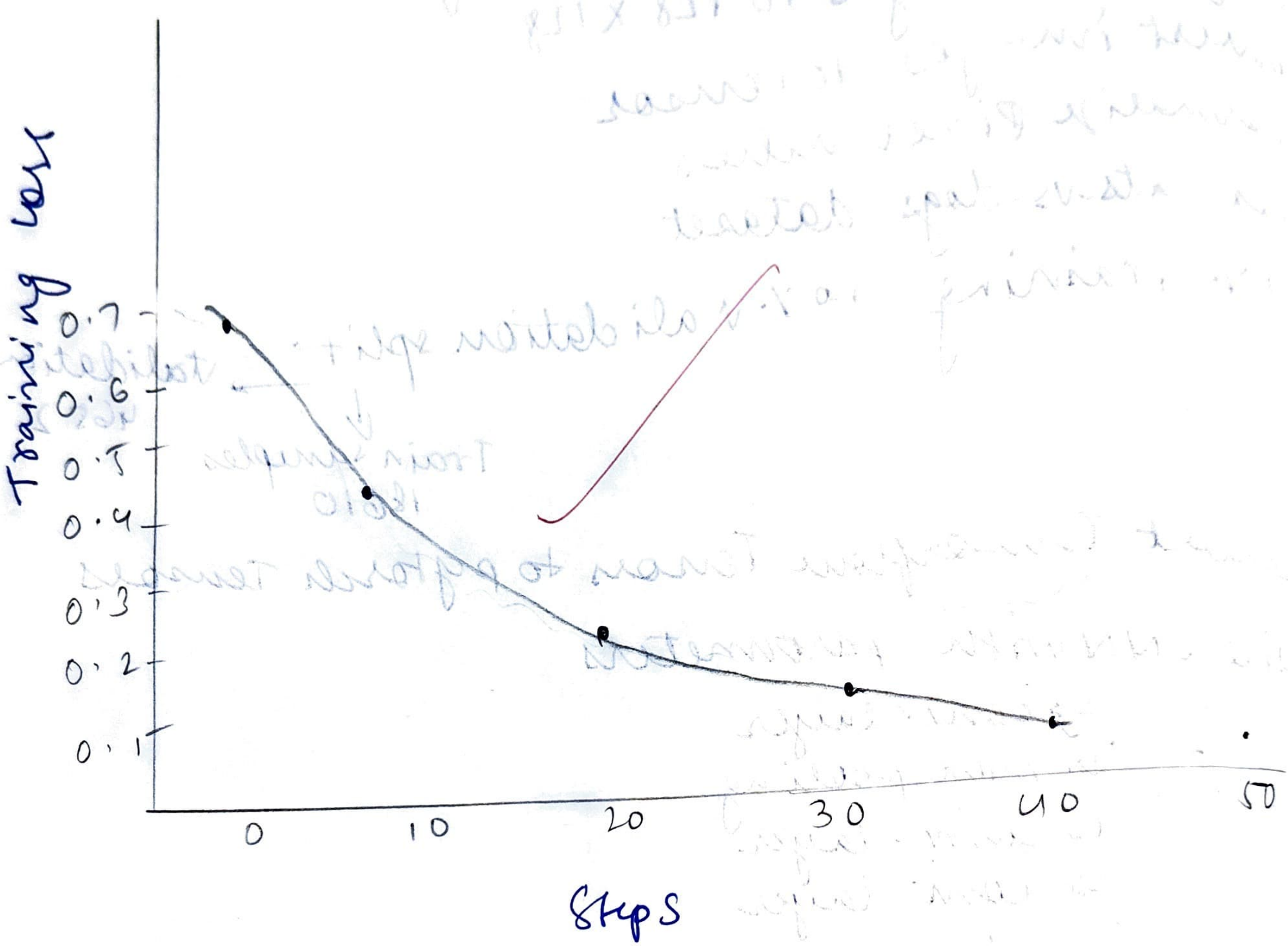
epoch [7/10] Avg Loss : 0.1326 Accuracy 94.74

epoch [8/10] Avg Loss : 0.1037 Accuracy : 95.92

epoch [9/10] Avg loss : 0.102 Accuray : 96.69

epoch [10/10] Avg loss : 0.101 accuracy : 97.10

Training loss over steps



- Use crossentropy loss funtion.
- To optimize model's trainable parameter use Adam optimizer
- Set no. of epochs
- Do forward and backward pass
- Print loss for 100 batches
- Plot the graph.
- End.

Result: Successfully built a CNN model.

23/9/25

```
dataset, info = tfds.load(
    'cats_vs_dogs',
    data_dir='/content/sample_data/cats_vs_dogs',
    as_supervised=True,
    with_info=True
)
```

```
import os

# List files and directories in the base data directory to understand the structure
data_dir_base = '/content/sample_data/cats_vs_dogs'
print(f"Contents of {data_dir_base}:")
print(os.listdir(data_dir_base))

# Assuming based on common tfds structures, there might be a version subdirectory
# Let's try listing the contents of potential subdirectories as well if found
for item in os.listdir(data_dir_base):
    item_path = os.path.join(data_dir_base, item)
    if os.path.isdir(item_path):
        print(f"\nContents of {item_path}:")
        try:
            print(os.listdir(item_path))
        except Exception as e:
            print(f"Could not list contents: {e}")
```

```
Contents of /content/sample_data/cats_vs_dogs:
['cats_vs_dogs', 'downloads']

Contents of /content/sample_data/cats_vs_dogs/cats_vs_dogs:
['4.0.1']

Contents of /content/sample_data/cats_vs_dogs/downloads:
['extracted', 'cats_vs_dogs']
```

11:32 AM

```
class CatsVsDogsDataset(Dataset):
    def __init__(self, data_dir, transform=None):
        self.data_dir = data_dir
        self.transform = transform
        # List files, filtering for jpg and ensuring they are files (not directories)
        self.image_files = [f for f in os.listdir(data_dir) if f.endswith('.jpg') and os.path.isfile(os.p


    def __len__(self):
        return len(self.image_files)

    def __getitem__(self, idx):
        img_name = os.path.join(self.data_dir, self.image_files[idx])
        image = Image.open(img_name).convert('RGB')
        label = 1 if 'cat' in self.image_files[idx] else 0

        if self.transform:
            image = self.transform(image)

        return image, label

# Define transforms
transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# Instantiate the datasets with the corrected data_dir path
corrected_data_dir = '/content/sample_data/cats_vs_dogs/cats_vs_dogs/4.0.1'

# Although tfds loaded with splits, the extracted data might be in a single directory.
# Let's confirm the number of images found in this directory.
# If all images are in one directory, we might need to split them manually for train/validation
# based on the file names or by using a SubsetRandomSampler later.
```

11:32 AM

```python
import os

# List contents of the suspected image directory
suspected_image_dir = '/content/sample_data/cats_vs_dogs/cats_vs_dogs/4.0.1'
print(f"Contents of {suspected_image_dir}:")

try:
    contents = os.listdir(suspected_image_dir)
    print(contents)

    # If there are subdirectories, list their contents too
    for item in contents:
        item_path = os.path.join(suspected_image_dir, item)
        if os.path.isdir(item_path):
            print(f"\nContents of {item_path}:")
            try:
                print(os.listdir(item_path))
            except Exception as e:
                print(f"Could not list contents: {e}")

except FileNotFoundError:
    print(f"Directory not found: {suspected_image_dir}")
except Exception as e:
    print(f"An error occurred while listing contents: {e}")
```

```
ontents of /content/sample_data/cats_vs_dogs/cats_vs_dogs/4.0.1:
'cats_vs_dogs-train.tfrecord-00014-of-00016', 'cats_vs_dogs-train.tfrecord-00006-of-00016',
rain.tfrecord-00012-of-00016', 'cats_vs_dogs-train.tfrecord-00000-of-00016', 'label.labels.
ogs-train.tfrecord-00013-of-00016', 'cats_vs_dogs-train.tfrecord-00011-of-00016', 'cats_vs
```

11:33 AM

```python
class TfDatasetToPyTorchIterableDataset(torch.utils.data.IterableDataset):
    def __init__(self, tf_dataset, transform=None):
        super().__init__()
        self.tf_dataset = tf_dataset
        self.transform = transform

    def __iter__(self):
        for image, label in self.tf_dataset:
            # Convert TensorFlow tensors to NumPy arrays, then to PyTorch tensors
            image = torch.from_numpy(image.numpy())
            label = torch.tensor(label.numpy())

            # Re-apply transform logic here to work with PyTorch tensor
            # Ensure image is float and has channels first for torchvision transforms
            image = image.permute(2, 0, 1).float() / 255.0 # Convert to channels first and normalize to [

            if self.transform:
                # Apply the rest of the transforms
                image = transforms.Compose([
                    transforms.Resize((128, 128)),
                    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
                ])(image) # Apply the rest of the transforms

            yield image, label

# Access the datasets using string keys as tfds.load with splits returns a dictionary
try:
    # Assuming the splits are named 'train' and 'validation' or similar.
    # Check the 'info' object or print the keys of 'dataset' to confirm the split names.
    # For this dataset, the default splits are 'train'. We'll use slicing for train/validation.
    train_tf_dataset = dataset['train'].take(int(info.splits['train'].num_examples * 0.8))
    validation_tf_dataset = dataset['train'].skip(int(info.splits['train'].num_examples * 0.8))

except (TypeError, KeyError):
```

11:33 AM

```
print(f"Batch size: {batch_size}")
print("DataLoaders are ready for iteration.")
```

```
Total number of examples in the training split: 18609
Total number of examples in the validation split: 4653
PyTorch IterableDatasets and DataLoaders created from split tf.data.Dataset.
Number of training examples: 18609
Number of validation examples: 4653
Batch size: 32
DataLoaders are ready for iteration.
```

4]:

```
import torch.nn as nn
import torch.nn.functional as F

class SimpleCNN(nn.Module):
    def __init__(self, num_classes=2):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.dropout = nn.Dropout(0.5)

        # Calculate the size of the flattened layer
        # Assuming input image size is 128x128 after transforms
        # After conv1 (128 -> 128), pool (128 -> 64)
```

11:33 AM

```
print("CNN model defined and instantiated.")
```

```
CNN model defined and instantiated.
```

15]:

```
import torch.nn as nn
import torch.optim as optim

# Define the loss function (Cross-Entropy Loss)
criterion = nn.CrossEntropyLoss()

# Define the optimizer (Adam)
# Assuming 'model' variable is available from the previous step
optimizer = optim.Adam(model.parameters(), lr=0.001)

print("Loss function (CrossEntropyLoss) and optimizer (Adam) defined.")
```

```
Loss function (CrossEntropyLoss) and optimizer (Adam) defined.
```

16]:

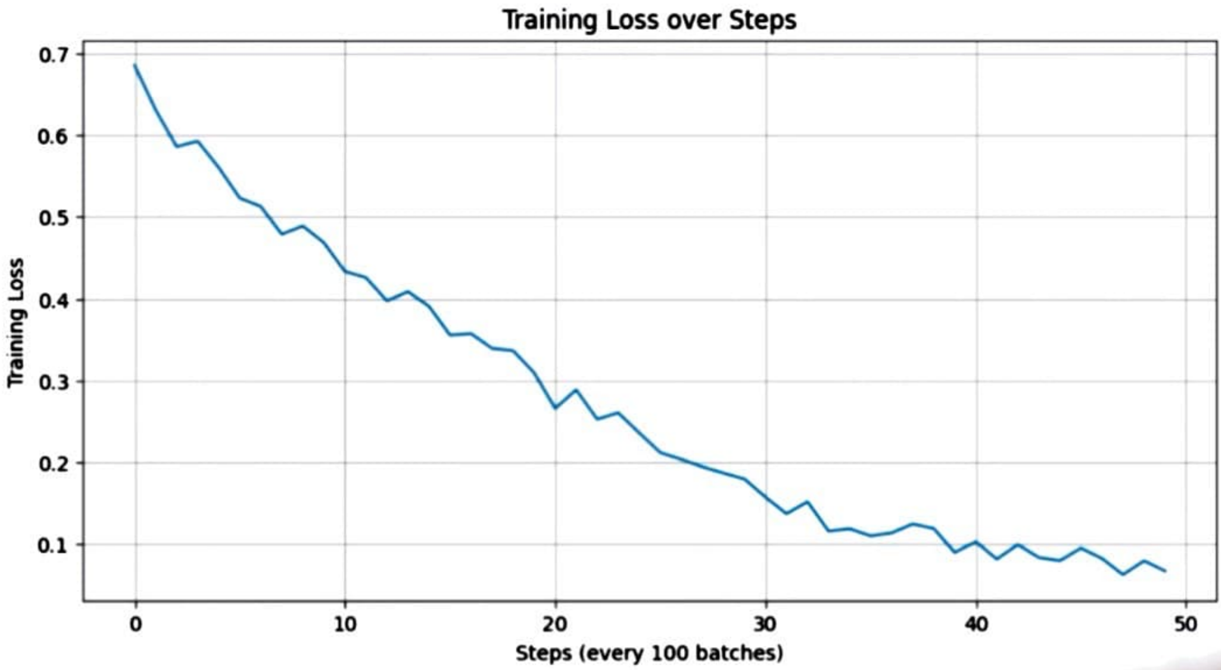
```
import torch

# Set the number of training epochs
num_epochs = 10

# Move model to the appropriate device (GPU if available)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

print(f"Training on device: {device}")

# Initialize a list to store training losses for plotting
training_losses = []

# Iterate through the specified number of epochs
```

11:33 AM

```
Epoch [1/10] finished. Average Loss: 0.5488, Accuracy: 66.65%
Epoch [2/10], Step [100/N/A (IterableDataset)], Loss: 0.5230
Epoch [2/10], Step [200/N/A (IterableDataset)], Loss: 0.5128
Epoch [2/10], Step [300/N/A (IterableDataset)], Loss: 0.4788
Epoch [2/10], Step [400/N/A (IterableDataset)], Loss: 0.4889
Epoch [2/10], Step [500/N/A (IterableDataset)], Loss: 0.4685
Epoch [2/10] finished. Average Loss: 0.4664, Accuracy: 76.30%
Epoch [3/10], Step [100/N/A (IterableDataset)], Loss: 0.4332
Epoch [3/10], Step [200/N/A (IterableDataset)], Loss: 0.4258
Epoch [3/10], Step [300/N/A (IterableDataset)], Loss: 0.3973
Epoch [3/10], Step [400/N/A (IterableDataset)], Loss: 0.4086
Epoch [3/10], Step [500/N/A (IterableDataset)], Loss: 0.3905
Epoch [3/10] finished. Average Loss: 0.3940, Accuracy: 81.39%
Epoch [4/10], Step [100/N/A (IterableDataset)], Loss: 0.3553
Epoch [4/10], Step [200/N/A (IterableDataset)], Loss: 0.3571
Epoch [4/10], Step [300/N/A (IterableDataset)], Loss: 0.3391
Epoch [4/10], Step [400/N/A (IterableDataset)], Loss: 0.3362
Epoch [4/10], Step [500/N/A (IterableDataset)], Loss: 0.3092
Epoch [4/10] finished. Average Loss: 0.3149, Accuracy: 85.28%
Epoch [5/10], Step [100/N/A (IterableDataset)], Loss: 0.2655
Epoch [5/10], Step [200/N/A (IterableDataset)], Loss: 0.2884
Epoch [5/10], Step [300/N/A (IterableDataset)], Loss: 0.2523
Epoch [5/10], Step [400/N/A (IterableDataset)], Loss: 0.2603
Epoch [5/10], Step [500/N/A (IterableDataset)], Loss: 0.2355
Epoch [5/10] finished. Average Loss: 0.2454, Accuracy: 89.20%
Epoch [6/10], Step [100/N/A (IterableDataset)], Loss: 0.2115
Epoch [6/10], Step [200/N/A (IterableDataset)], Loss: 0.2033
Epoch [6/10], Step [300/N/A (IterableDataset)], Loss: 0.1939
Epoch [6/10], Step [400/N/A (IterableDataset)], Loss: 0.1863
Epoch [6/10], Step [500/N/A (IterableDataset)], Loss: 0.1791
Epoch [6/10] finished. Average Loss: 0.1801, Accuracy: 92.23%
Epoch [7/10], Step [100/N/A (IterableDataset)], Loss: 0.1570
Epoch [7/10], Step [200/N/A (IterableDataset)], Loss: 0.1367
Epoch [7/10], Step [300/N/A (IterableDataset)], Loss: 0.1514
Epoch [7/10], Step [400/N/A (IterableDataset)], Loss: 0.1154
Epoch [7/10], Step [500/N/A (IterableDataset)], Loss: 0.1183
```

11:33 AM

```
    plt.grid(True)
    plt.show()
else:
    print("Training losses were not recorded. Please modify the training loop to store loss values.")
```



11:34 AM