Convolutional Neural Network

Group No :151

Group Member Names:

Peyala Samarasimha Reddy - 2023AA05072 100% Contribution

Monisha G - 2023AA05536 100% Contribution

Akshay Mohan - 2023AA05315 100% Contribution

Sreelakshmi Ajith - 2023AA05316 100% Contribution

In [14]:

```
!pip uninstall tensorflow
!pip install tensorflow
```

```
Found existing installation: tensorflow 2.17.0
Uninstalling tensorflow-2.17.0:
  Would remove:
    /usr/local/bin/import_pb_to_tensorboard
    /usr/local/bin/saved_model_cli
    /usr/local/bin/tensorboard
    /usr/local/bin/tf_upgrade_v2
    /usr/local/bin/tflite_convert
    /usr/local/bin/toco
    /usr/local/bin/toco_from_protos
    /usr/local/lib/python3.10/dist-packages/tensorflow-2.17.0.dist-info/*
    /usr/local/lib/python3.10/dist-packages/tensorflow/*
Proceed (Y/n)? y
Y
  Successfully uninstalled tensorflow-2.17.0
Collecting tensorflow
  Using cached tensorflow-2.17.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl.metadata (4.2 kB)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packag
es (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-pac
kages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/p
ython3.10/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-p
ackages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-packages
(from tensorflow) (3.11.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-pack
ages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/di
st-packages (from tensorflow) (0.4.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-pac
kages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (f
rom tensorflow) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=
4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow)
(3.20.3)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-p
ackages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages
(from tensorflow) (71.0.4)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages
(from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-pack
ages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/d
ist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-package
s (from tensorflow) (1.16.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-p
ackages (from tensorflow) (1.64.1)
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/di
st-packages (from tensorflow) (2.17.0)
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-packages
(from tensorflow) (3.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/
python3.10/dist-packages (from tensorflow) (0.37.1)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (1.26.4)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-pa
ckages (from astunparse>=1.6.0->tensorflow) (0.44.0)
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from k
eras>=3.2.0->tensorflow) (13.8.1)
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from
keras>=3.2.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from
keras>=3.2.0->tensorflow) (0.12.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/d
ist-packages (from requests<3,>=2.21.0->tensorflow) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
(from requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
ckages (from requests<3,>=2.21.0->tensorflow) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pa
ckages (from requests<3,>=2.21.0->tensorflow) (2024.8.30)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packa
ges (from tensorboard<2.18,>=2.17->tensorflow) (3.7)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/li
b/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packa
ges (from tensorboard<2.18,>=2.17->tensorflow) (3.0.4)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-pac
kages (from werkzeug>=1.0.1->tensorboard<2.18,>=2.17->tensorflow) (2.1.5)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist
-packages (from rich->keras>=3.2.0->tensorflow) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/di
st-packages (from rich->keras>=3.2.0->tensorflow) (2.18.0)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages
(from markdown-it-py>=2.2.0->rich->keras>=3.2.0->tensorflow) (0.1.2)
Using cached tensorflow-2.17.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.
whl (601.3 MB)
Installing collected packages: tensorflow
Successfully installed tensorflow-2.17.0
```

In [1]:

```
import tensorflow as tf
print(tf.__version__)
```

```
2.17.0
```

In [2]:

```
# Import Libraries
# - Tensorflow
# - Keras
# - numpy and random

import tensorflow as tf
from tensorflow.keras import models
from tensorflow.keras import layers

import random
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
```

In [4]:

```
random.seed(42)         # Initialize the random number generator.
np.random.seed(42)      # With the seed reset, the same set of numbers will appear ev
tf.random.set_seed(42)   # sets the graph-level random seed
```

Dataset - MNIST

In [5]:

```
# Use the MNIST dataset  of Keras.

mnist = tf.keras.datasets.mnist

(Xtrain, Ytrain), (Xtest,Ytest) = mnist.load_data()
```

```
# Display size of dataset
Xtrain = Xtrain.reshape((60000,28,28,1))
Xtrain = Xtrain.astype('float32')/255

Xtest = Xtest.reshape((10000,28,28,1))
Xtest = Xtest.astype('float32')/255

Ytrain = tf.keras.utils.to_categorical(Ytrain)
Ytest = tf.keras.utils.to_categorical(Ytest)

print(Xtrain.shape, Xtest.shape)
print(Ytrain.shape, Ytest.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnis
t.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
(60000, 28, 28, 1) (10000, 28, 28, 1)
(60000, 10) (10000, 10)
```

Create a CNN Model

In [6]:

```
# Create a Sequential model object
cnnModel = models.Sequential()

# Add layers Conv2D for CNN and specify MaxPooling

# Layer 1 = input layer
cnnModel.add(layers.Conv2D(32, (3,3), activation="relu", input_shape=(28,28,1) ))

cnnModel.add(layers.MaxPooling2D((2,2)))


# Layer 2
cnnModel.add(layers.Conv2D(64, (3,3), activation="relu"))

cnnModel.add(layers.MaxPooling2D((2,2)))


# Layer 3
cnnModel.add(layers.Conv2D(64, (3,3), activation="relu" ))

cnnModel.add(layers.Flatten())


# Add Dense layers or fully connected layers
# Layer 4
cnnModel.add(layers.Dense(64, activation="relu" ))

# Layer 5
cnnModel.add(layers.Dense(32, activation="relu" ))

# Layer 6
cnnModel.add(layers.Dense(10, activation="softmax" ))

cnnModel.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:10
7: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When usi
ng 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"

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━
┃ Layer (type)                    ┃ Output Shape           ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━
│ conv2d (Conv2D)                 │ (None, 26, 26, 32)     │
├─────────────────────────────────┼────────────────────────┼─────────
│ max_pooling2d (MaxPooling2D)    │ (None, 13, 13, 32)     │
├─────────────────────────────────┼────────────────────────┼─────────
│ conv2d_1 (Conv2D)               │ (None, 11, 11, 64)     │
├─────────────────────────────────┼────────────────────────┼─────────
│ max_pooling2d_1 (MaxPooling2D)  │ (None, 5, 5, 64)       │
├─────────────────────────────────┼────────────────────────┼─────────
│ conv2d_2 (Conv2D)               │ (None, 3, 3, 64)       │
├─────────────────────────────────┼────────────────────────┼─────────
│ flatten (Flatten)               │ (None, 576)            │
├─────────────────────────────────┼────────────────────────┼─────────
│ dense (Dense)                   │ (None, 64)             │
├─────────────────────────────────┼────────────────────────┼─────────
│ dense_1 (Dense)                 │ (None, 32)             │
├─────────────────────────────────┼────────────────────────┼─────────
│ dense_2 (Dense)                 │ (None, 10)             │
└─────────────────────────────────┴────────────────────────┴─────────
 Total params: 95,082 (371.41 KB)
 Trainable params: 95,082 (371.41 KB)
 Non-trainable params: 0 (0.00 B)
```

In [7]:

```
# Configure  the model for training, by using appropriate optimizers and regularizati
# Available optimizer: adam, rmsprop, adagrad, sgd
# loss:  objective that the model will try to minimize.
# Available loss: categorical_crossentropy, binary_crossentropy, mean_squared_error
# metrics: List of metrics to be evaluated by the model during training and testing.

cnnModel.compile(optimizer = "adam", loss = "categorical_crossentropy", metrics = ["a
```

In [8]:

```
# train the model

history = cnnModel.fit(Xtrain, Ytrain, epochs = 25, batch_size = 64, validation_split
Epoch 1/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 53s 61ms/step - accuracy: 0.8367 - loss: 0.5159 - val_acc
uracy: 0.9810 - val_loss: 0.0624
Epoch 2/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 83s 62ms/step - accuracy: 0.9802 - loss: 0.0636 - val_acc
uracy: 0.9862 - val_loss: 0.0500
Epoch 3/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 80s 60ms/step - accuracy: 0.9874 - loss: 0.0409 - val_acc
uracy: 0.9862 - val_loss: 0.0525
Epoch 4/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 82s 60ms/step - accuracy: 0.9900 - loss: 0.0320 - val_acc
uracy: 0.9887 - val_loss: 0.0441
Epoch 5/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 53s 62ms/step - accuracy: 0.9919 - loss: 0.0246 - val_acc
uracy: 0.9875 - val_loss: 0.0483
Epoch 6/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 79s 59ms/step - accuracy: 0.9934 - loss: 0.0216 - val_acc
uracy: 0.9888 - val_loss: 0.0490
Epoch 7/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 83s 60ms/step - accuracy: 0.9946 - loss: 0.0170 - val_acc
uracy: 0.9893 - val_loss: 0.0443
Epoch 8/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 81s 60ms/step - accuracy: 0.9945 - loss: 0.0159 - val_acc
uracy: 0.9903 - val_loss: 0.0410
Epoch 9/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 83s 60ms/step - accuracy: 0.9952 - loss: 0.0140 - val_acc
uracy: 0.9893 - val_loss: 0.0444
Epoch 10/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 50s 59ms/step - accuracy: 0.9969 - loss: 0.0093 - val_acc
uracy: 0.9895 - val_loss: 0.0444
Epoch 11/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 82s 59ms/step - accuracy: 0.9969 - loss: 0.0090 - val_acc
uracy: 0.9883 - val_loss: 0.0563
Epoch 12/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 49s 58ms/step - accuracy: 0.9965 - loss: 0.0107 - val_acc
uracy: 0.9915 - val_loss: 0.0412
Epoch 13/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 86s 63ms/step - accuracy: 0.9977 - loss: 0.0067 - val_acc
uracy: 0.9902 - val_loss: 0.0516
Epoch 14/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 79s 60ms/step - accuracy: 0.9969 - loss: 0.0087 - val_acc
uracy: 0.9890 - val_loss: 0.0614
Epoch 15/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 81s 59ms/step - accuracy: 0.9981 - loss: 0.0050 - val_acc
uracy: 0.9898 - val_loss: 0.0563
Epoch 16/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 82s 59ms/step - accuracy: 0.9970 - loss: 0.0100 - val_acc
uracy: 0.9915 - val_loss: 0.0488
Epoch 17/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 50s 59ms/step - accuracy: 0.9982 - loss: 0.0056 - val_acc
uracy: 0.9927 - val_loss: 0.0449
Epoch 18/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 51s 61ms/step - accuracy: 0.9982 - loss: 0.0056 - val_acc
uracy: 0.9913 - val_loss: 0.0558
Epoch 19/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 82s 60ms/step - accuracy: 0.9981 - loss: 0.0060 - val_acc
uracy: 0.9908 - val_loss: 0.0499
Epoch 20/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 81s 59ms/step - accuracy: 0.9986 - loss: 0.0041 - val_acc
uracy: 0.9897 - val_loss: 0.0595
Epoch 21/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 87s 65ms/step - accuracy: 0.9986 - loss: 0.0043 - val_acc
uracy: 0.9905 - val_loss: 0.0573
Epoch 22/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 78s 60ms/step - accuracy: 0.9985 - loss: 0.0039 - val_acc
uracy: 0.9905 - val_loss: 0.0575
```

```
Epoch 23/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 81s 60ms/step - accuracy: 0.9987 - loss: 0.0043 - val_acc
uracy: 0.9898 - val_loss: 0.0573
Epoch 24/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 81s 59ms/step - accuracy: 0.9983 - loss: 0.0059 - val_acc
uracy: 0.9900 - val_loss: 0.0716
Epoch 25/25
844/844 ━━━━━━━━━━━━━━━━━━━━ 82s 59ms/step - accuracy: 0.9973 - loss: 0.0083 - val_acc
uracy: 0.9925 - val_loss: 0.0496
```

In [10]:

```
print('Final training loss \t', history.history['loss'][-1])
print('Final training accuracy ', history.history['accuracy'][-1])
```

```
Final training loss      0.005073322914540768
Final training accuracy  0.9984074234962463
```

Results and Outputs

In [11]:

```
# testing the model

testLoss, testAccuracy = cnnModel.evaluate( Xtest, Ytest)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.9885 - loss: 0.0761
```

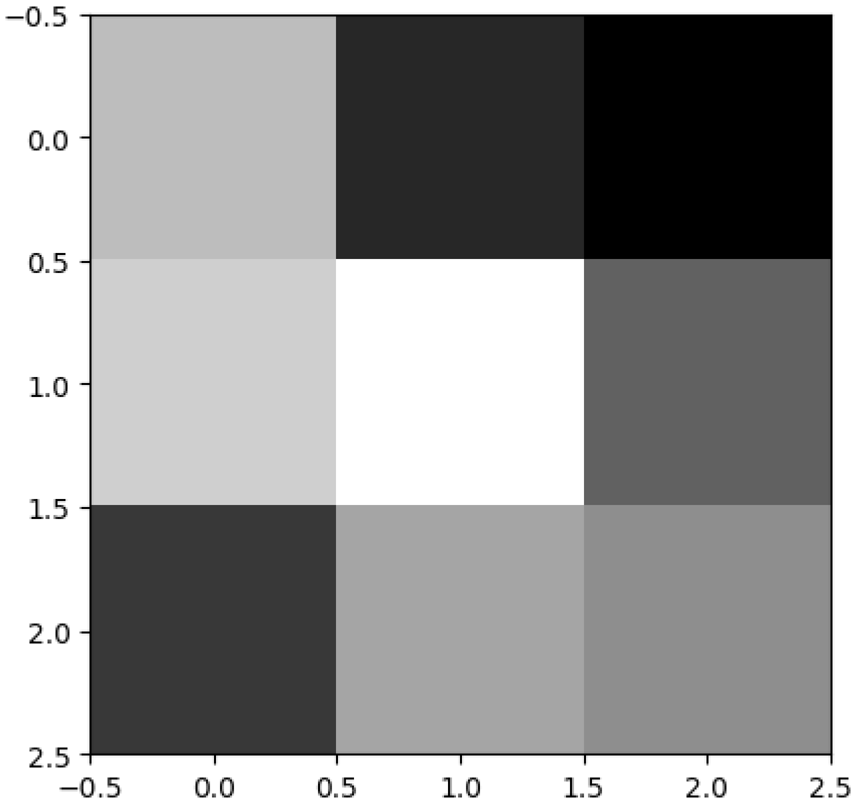
In [12]:

```
print('Testing loss \t', testLoss)
print('Testing accuracy ', testAccuracy)
```

```
Testing loss     0.05195753276348114
Testing accuracy  0.9919999837875366
```

In [13]:

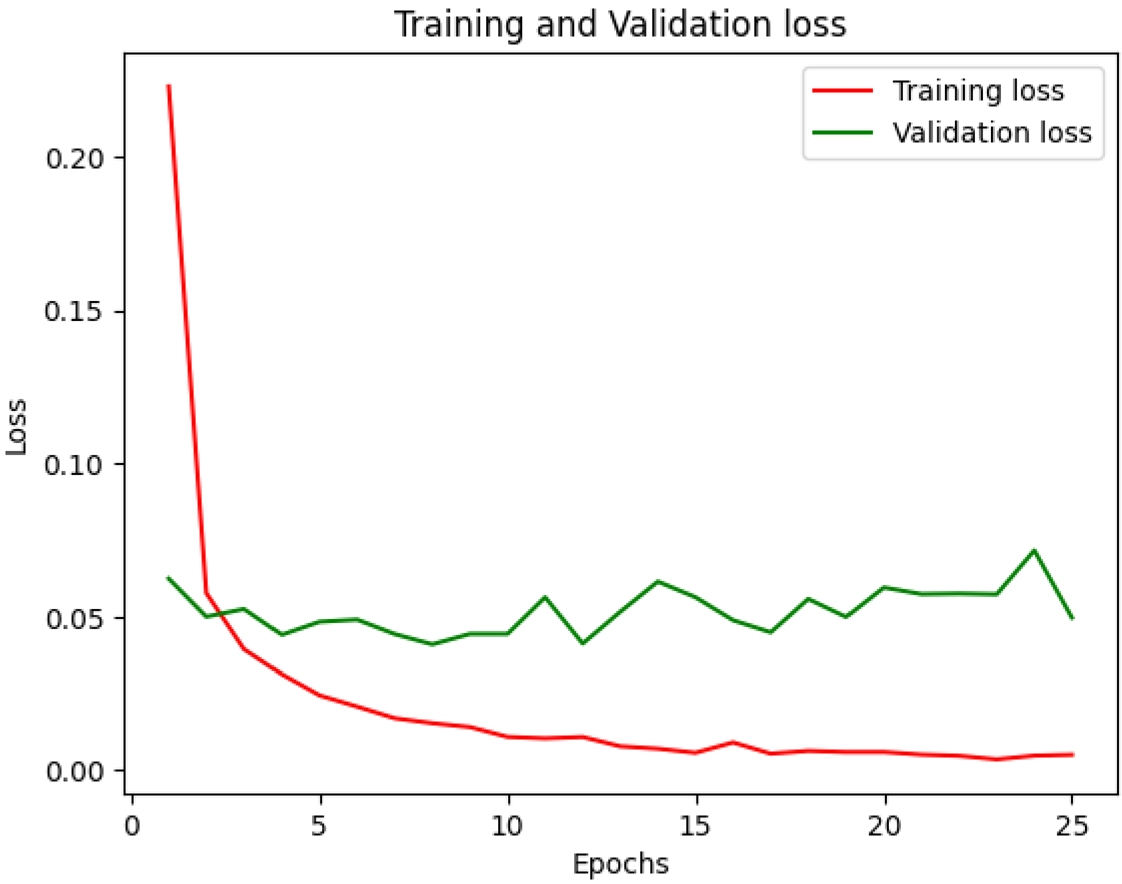
```
# shows the weights in layer 0 conv2d as gray map
top_layer = cnnModel.layers[0]
plt.imshow(top_layer.get_weights()[0][:, :, :, 0].squeeze(), cmap='gray')
plt.show()
```

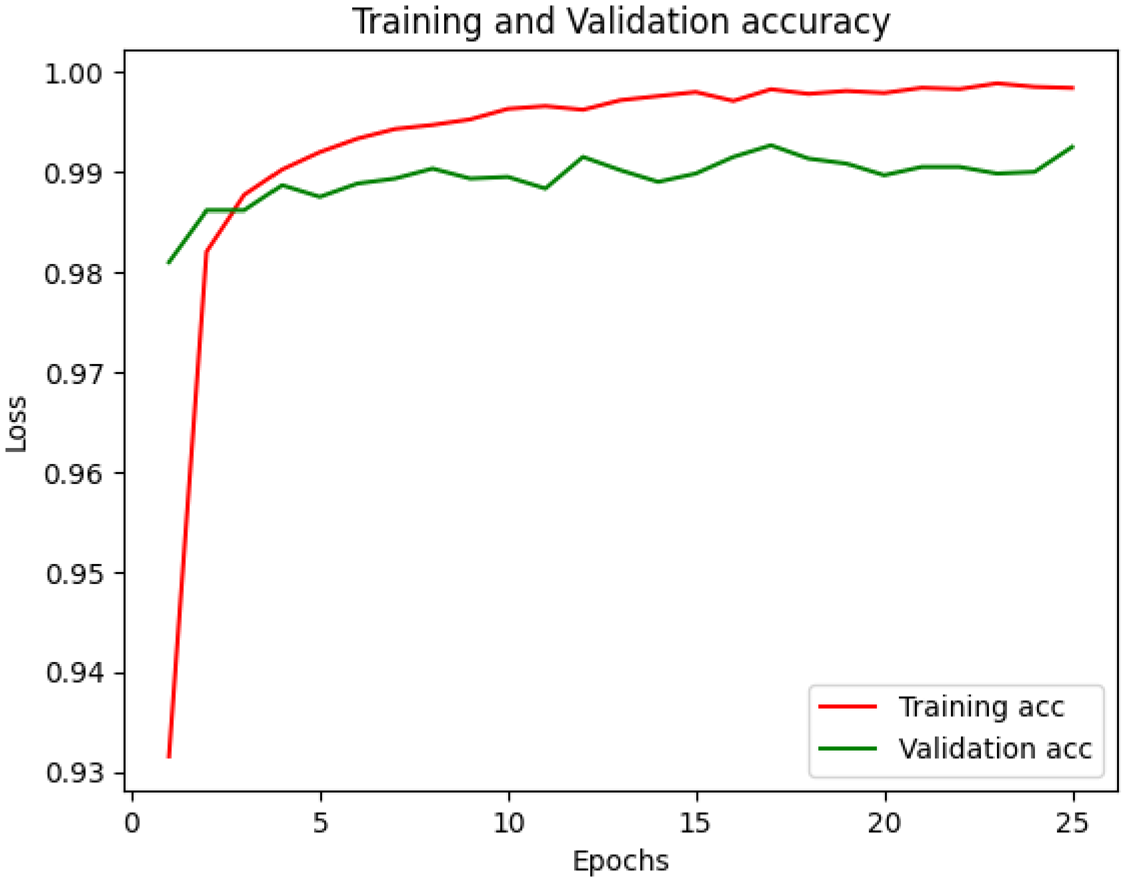


In [16]:

```
# plotting training and validation loss

loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
```

```
plt.plot(epochs, loss, color='red', label='Training loss')
plt.plot(epochs, val_loss, color='green', label='Validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# plotting training and validation accuracy

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
plt.plot(epochs, acc, color='red', label='Training acc')
plt.plot(epochs, val_acc, color='green', label='Validation acc')
plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```





Confusion Matrix generation

Prediction for a specific testing data generte confusion matrix

In [17]:

```
Y_prediction = cnnModel.predict(Xtest)

# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_prediction, axis = 1)

# Convert validation observations to one hot vectors
Y_true = np.argmax(Ytest,axis = 1)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step
```

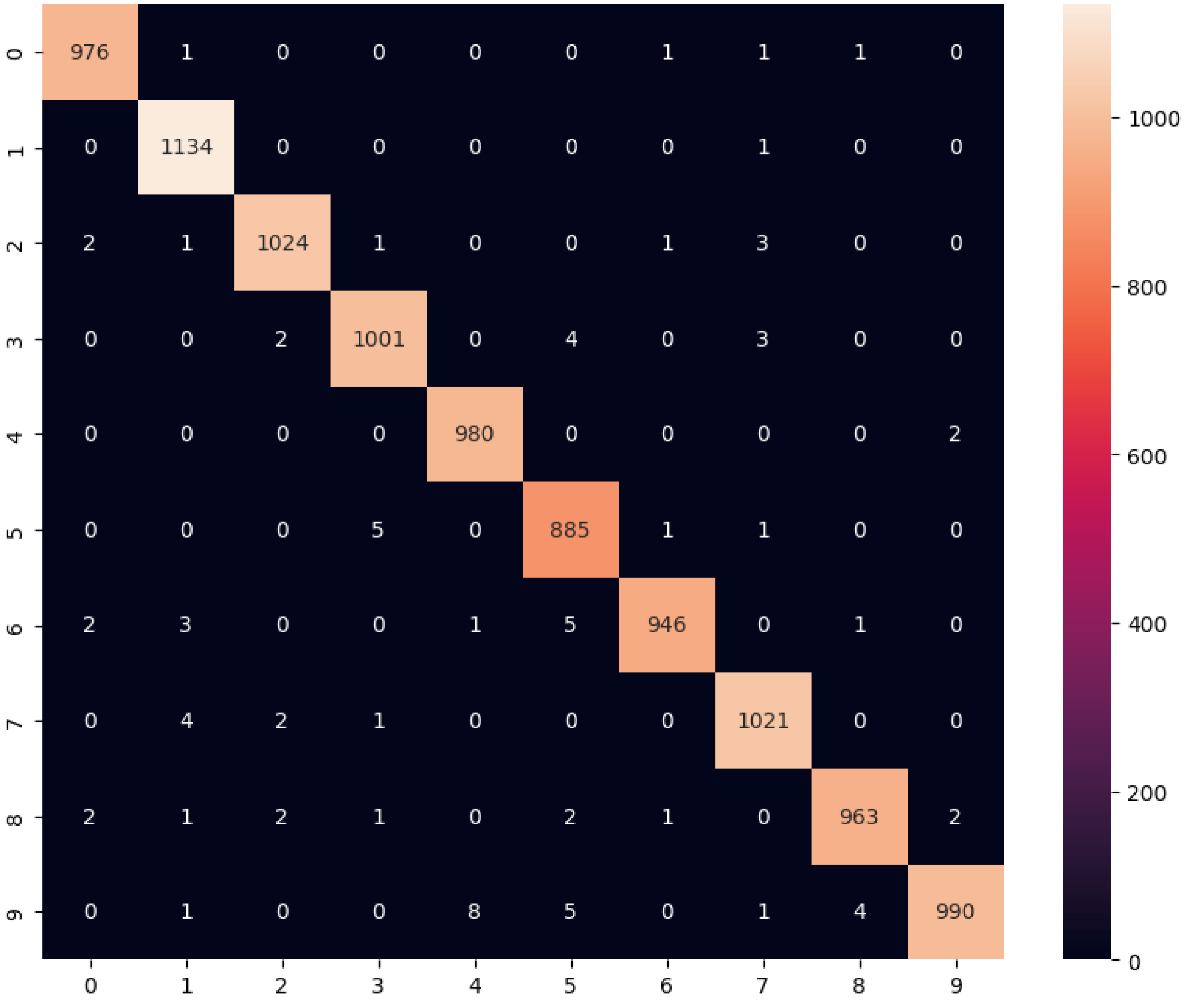
In [18]:

```
# Classification Report

from sklearn.metrics import classification_report

print(classification_report(Y_true, Y_pred_classes))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       980
           1       0.99      1.00      0.99      1135
           2       0.99      0.99      0.99      1032
           3       0.99      0.99      0.99      1010
           4       0.99      1.00      0.99       982
           5       0.98      0.99      0.99       892
           6       1.00      0.99      0.99       958
           7       0.99      0.99      0.99      1028
           8       0.99      0.99      0.99       974
           9       1.00      0.98      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

In [19]:

```python
# confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns

# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)

plt.figure(figsize=(10,8))
sns.heatmap(confusion_mtx, annot=True, fmt="d");
```



	0	1	2	3	4	5	6	7	8	9
0	976	1	0	0	0	0	1	1	1	0
1	0	1134	0	0	0	0	0	1	0	0
2	2	1	1024	1	0	0	1	3	0	0
3	0	0	2	1001	0	4	0	3	0	0
4	0	0	0	0	980	0	0	0	0	2
5	0	0	0	5	0	885	1	1	0	0
6	2	3	0	0	1	5	946	0	1	0
7	0	4	2	1	0	0	0	1021	0	0
8	2	1	2	1	0	2	1	0	963	2
9	0	1	0	0	8	5	0	1	4	990

Modify the code to get a better testing

accuracy.

- Change the number of hidden units
- Increase the number of hidden layers
- Use a different optimizer
- Train for more epochs for better graphs
- Try using CIFAR dataset

CIFAR-10 Image Classification using a 3-Layer CNN with 256 and 128 Hidden Units and RMSprop Optimizer

In [20]:

```
# Importing required libraries
import numpy as np
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

# One-hot encoding the labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Model building
model = models.Sequential()

# Add first convolutional layer
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))

# Add second convolutional layer
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Add third convolutional layer
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Flattening the results
model.add(layers.Flatten())

# Add first dense layer (hidden units increased to 256)
model.add(layers.Dense(256, activation='relu'))

# Add second dense layer to increase hidden layers
model.add(layers.Dense(128, activation='relu'))

# Add output layer
model.add(layers.Dense(10, activation='softmax'))

cnnModel.summary()
```

```python

# Compile the model using RMSprop optimizer instead of Adam
model.compile(optimizer='RMSprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model (increase epochs to 50)
history = model.fit(x_train, y_train, epochs=50, validation_data=(x_test, y_test))

# Plotting training and validation accuracy over epochs
```

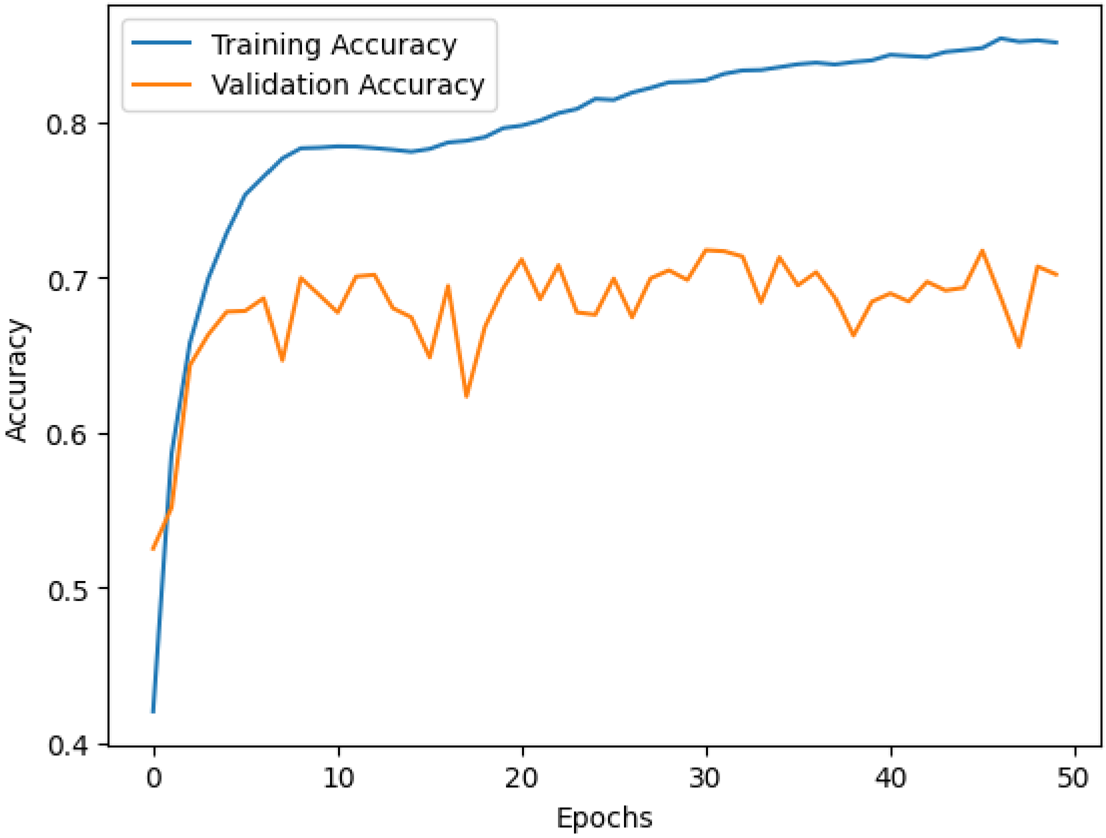
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 11s 0us/step
```

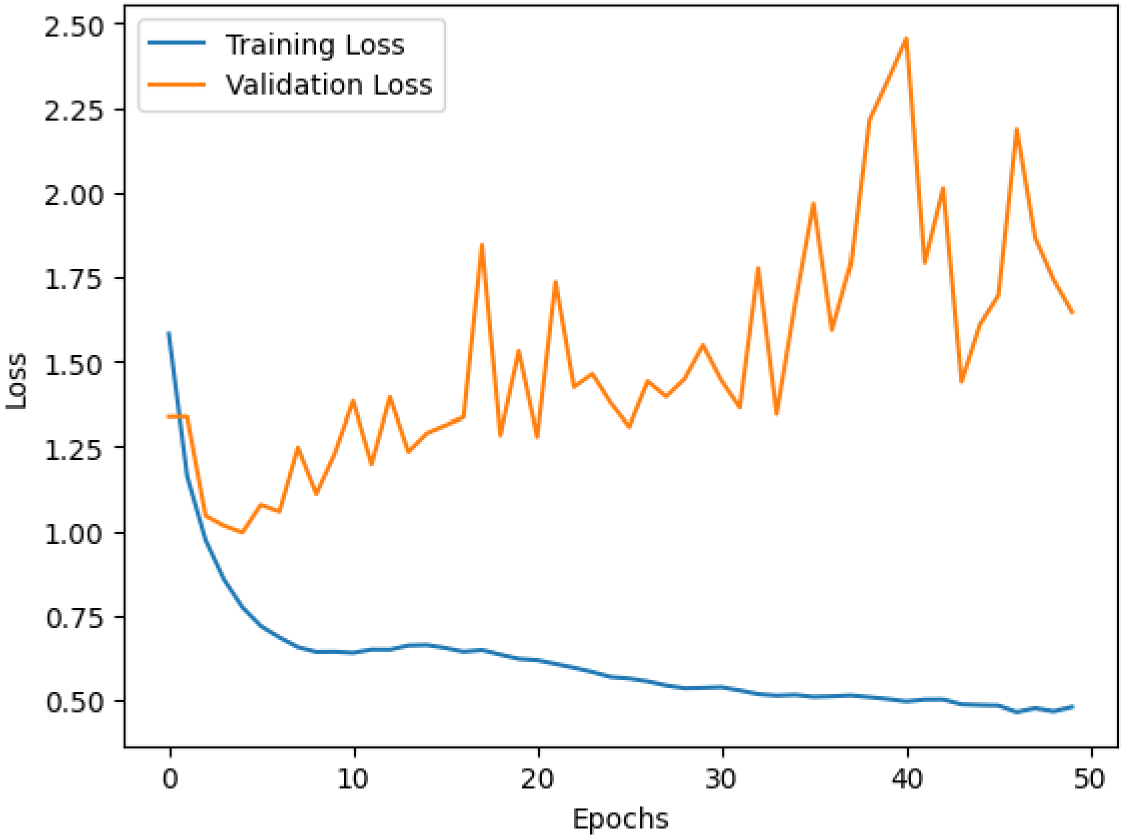
```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:10
7: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When usi
ng Sequential models, prefer using an `Input(shape)` object as the first layer in the
model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 77s 48ms/step - accuracy: 0.3224 - loss: 1.8183 - val_a
ccuracy: 0.5253 - val_loss: 1.3370
Epoch 2/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 77s 49ms/step - accuracy: 0.5640 - loss: 1.2252 - val_a
ccuracy: 0.5517 - val_loss: 1.3373
Epoch 3/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 80s 48ms/step - accuracy: 0.6434 - loss: 1.0145 - val_a
ccuracy: 0.6436 - val_loss: 1.0460
Epoch 4/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 76s 49ms/step - accuracy: 0.6898 - loss: 0.8838 - val_a
ccuracy: 0.6634 - val_loss: 1.0166
Epoch 5/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 75s 48ms/step - accuracy: 0.7209 - loss: 0.7969 - val_a
ccuracy: 0.6781 - val_loss: 0.9968
Epoch 6/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 78s 50ms/step - accuracy: 0.7474 - loss: 0.7358 - val_a
ccuracy: 0.6786 - val_loss: 1.0786
Epoch 7/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 78s 48ms/step - accuracy: 0.7614 - loss: 0.6970 - val_a
ccuracy: 0.6868 - val_loss: 1.0583
Epoch 8/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 48ms/step - accuracy: 0.7743 - loss: 0.6642 - val_a
ccuracy: 0.6467 - val_loss: 1.2481
Epoch 9/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 48ms/step - accuracy: 0.7799 - loss: 0.6495 - val_a
ccuracy: 0.6999 - val_loss: 1.1105
Epoch 10/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 76s 49ms/step - accuracy: 0.7825 - loss: 0.6505 - val_a
ccuracy: 0.6889 - val_loss: 1.2301
Epoch 11/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 48ms/step - accuracy: 0.7821 - loss: 0.6439 - val_a
ccuracy: 0.6776 - val_loss: 1.3845
Epoch 12/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 48ms/step - accuracy: 0.7828 - loss: 0.6551 - val_a
ccuracy: 0.7008 - val_loss: 1.1982
Epoch 13/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 48ms/step - accuracy: 0.7825 - loss: 0.6468 - val_a
ccuracy: 0.7019 - val_loss: 1.3955
Epoch 14/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 83s 49ms/step - accuracy: 0.7814 - loss: 0.6576 - val_a
ccuracy: 0.6804 - val_loss: 1.2349
Epoch 15/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 77s 49ms/step - accuracy: 0.7803 - loss: 0.6661 - val_a
ccuracy: 0.6745 - val_loss: 1.2898
Epoch 16/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 49ms/step - accuracy: 0.7804 - loss: 0.6608 - val_a
ccuracy: 0.6486 - val_loss: 1.3129
Epoch 17/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 48ms/step - accuracy: 0.7857 - loss: 0.6500 - val_a
ccuracy: 0.6948 - val_loss: 1.3359
Epoch 18/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 77s 49ms/step - accuracy: 0.7876 - loss: 0.6496 - val_a
ccuracy: 0.6234 - val_loss: 1.8449
Epoch 19/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 48ms/step - accuracy: 0.7895 - loss: 0.6391 - val_a
ccuracy: 0.6683 - val_loss: 1.2840
Epoch 20/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 48ms/step - accuracy: 0.7935 - loss: 0.6277 - val_a
ccuracy: 0.6933 - val_loss: 1.5315
Epoch 21/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 76s 49ms/step - accuracy: 0.7960 - loss: 0.6315 - val_a
ccuracy: 0.7118 - val_loss: 1.2789
Epoch 22/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 75s 48ms/step - accuracy: 0.7992 - loss: 0.6154 - val_a
ccuracy: 0.6862 - val_loss: 1.7356
```

```
Epoch 23/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 77s 49ms/step - accuracy: 0.8045 - loss: 0.6002 - val_a
ccuracy: 0.7081 - val_loss: 1.4250
Epoch 24/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 75s 48ms/step - accuracy: 0.8102 - loss: 0.5829 - val_a
ccuracy: 0.6777 - val_loss: 1.4634
Epoch 25/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 48ms/step - accuracy: 0.8147 - loss: 0.5692 - val_a
ccuracy: 0.6762 - val_loss: 1.3776
Epoch 26/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 77s 49ms/step - accuracy: 0.8139 - loss: 0.5631 - val_a
ccuracy: 0.6995 - val_loss: 1.3082
Epoch 27/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 48ms/step - accuracy: 0.8203 - loss: 0.5524 - val_a
ccuracy: 0.6745 - val_loss: 1.4419
Epoch 28/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 48ms/step - accuracy: 0.8205 - loss: 0.5485 - val_a
ccuracy: 0.6996 - val_loss: 1.3966
Epoch 29/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 48ms/step - accuracy: 0.8255 - loss: 0.5387 - val_a
ccuracy: 0.7048 - val_loss: 1.4485
Epoch 30/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 48ms/step - accuracy: 0.8264 - loss: 0.5332 - val_a
ccuracy: 0.6986 - val_loss: 1.5488
Epoch 31/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 77s 49ms/step - accuracy: 0.8279 - loss: 0.5404 - val_a
ccuracy: 0.7178 - val_loss: 1.4441
Epoch 32/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 49ms/step - accuracy: 0.8312 - loss: 0.5283 - val_a
ccuracy: 0.7171 - val_loss: 1.3639
Epoch 33/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 49ms/step - accuracy: 0.8321 - loss: 0.5244 - val_a
ccuracy: 0.7137 - val_loss: 1.7766
Epoch 34/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 78s 50ms/step - accuracy: 0.8329 - loss: 0.5138 - val_a
ccuracy: 0.6841 - val_loss: 1.3450
Epoch 35/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 76s 48ms/step - accuracy: 0.8354 - loss: 0.5096 - val_a
ccuracy: 0.7132 - val_loss: 1.6694
Epoch 36/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 78s 50ms/step - accuracy: 0.8382 - loss: 0.5081 - val_a
ccuracy: 0.6951 - val_loss: 1.9676
Epoch 37/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 50ms/step - accuracy: 0.8396 - loss: 0.5070 - val_a
ccuracy: 0.7036 - val_loss: 1.5931
Epoch 38/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 80s 48ms/step - accuracy: 0.8350 - loss: 0.5211 - val_a
ccuracy: 0.6874 - val_loss: 1.7886
Epoch 39/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 80s 47ms/step - accuracy: 0.8377 - loss: 0.5174 - val_a
ccuracy: 0.6627 - val_loss: 2.2141
Epoch 40/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 47ms/step - accuracy: 0.8416 - loss: 0.4940 - val_a
ccuracy: 0.6846 - val_loss: 2.3317
Epoch 41/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 80s 51ms/step - accuracy: 0.8431 - loss: 0.4989 - val_a
ccuracy: 0.6899 - val_loss: 2.4552
Epoch 42/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 76s 47ms/step - accuracy: 0.8413 - loss: 0.5060 - val_a
ccuracy: 0.6847 - val_loss: 1.7924
Epoch 43/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 74s 47ms/step - accuracy: 0.8407 - loss: 0.5057 - val_a
ccuracy: 0.6974 - val_loss: 2.0131
Epoch 44/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 83s 48ms/step - accuracy: 0.8435 - loss: 0.4985 - val_a
ccuracy: 0.6918 - val_loss: 1.4399
```

```
Epoch 45/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 48ms/step - accuracy: 0.8493 - loss: 0.4717 - val_a
ccuracy: 0.6936 - val_loss: 1.6100
Epoch 46/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 47ms/step - accuracy: 0.8493 - loss: 0.4778 - val_a
ccuracy: 0.7175 - val_loss: 1.6955
Epoch 47/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 47ms/step - accuracy: 0.8561 - loss: 0.4688 - val_a
ccuracy: 0.6873 - val_loss: 2.1864
Epoch 48/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 47ms/step - accuracy: 0.8533 - loss: 0.4695 - val_a
ccuracy: 0.6555 - val_loss: 1.8660
Epoch 49/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 75s 48ms/step - accuracy: 0.8546 - loss: 0.4620 - val_a
ccuracy: 0.7073 - val_loss: 1.7421
Epoch 50/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 48ms/step - accuracy: 0.8512 - loss: 0.4787 - val_a
ccuracy: 0.7022 - val_loss: 1.6467
```





```
313/313 - 4s - 12ms/step - accuracy: 0.7022 - loss: 1.6467
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.sa
ving.save_model(model)`. This file format is considered legacy. We recommend using ins
tead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.sav
e_model(model, 'my_model.keras')`.
Test accuracy: 0.7021999955177307
```

In [22]:

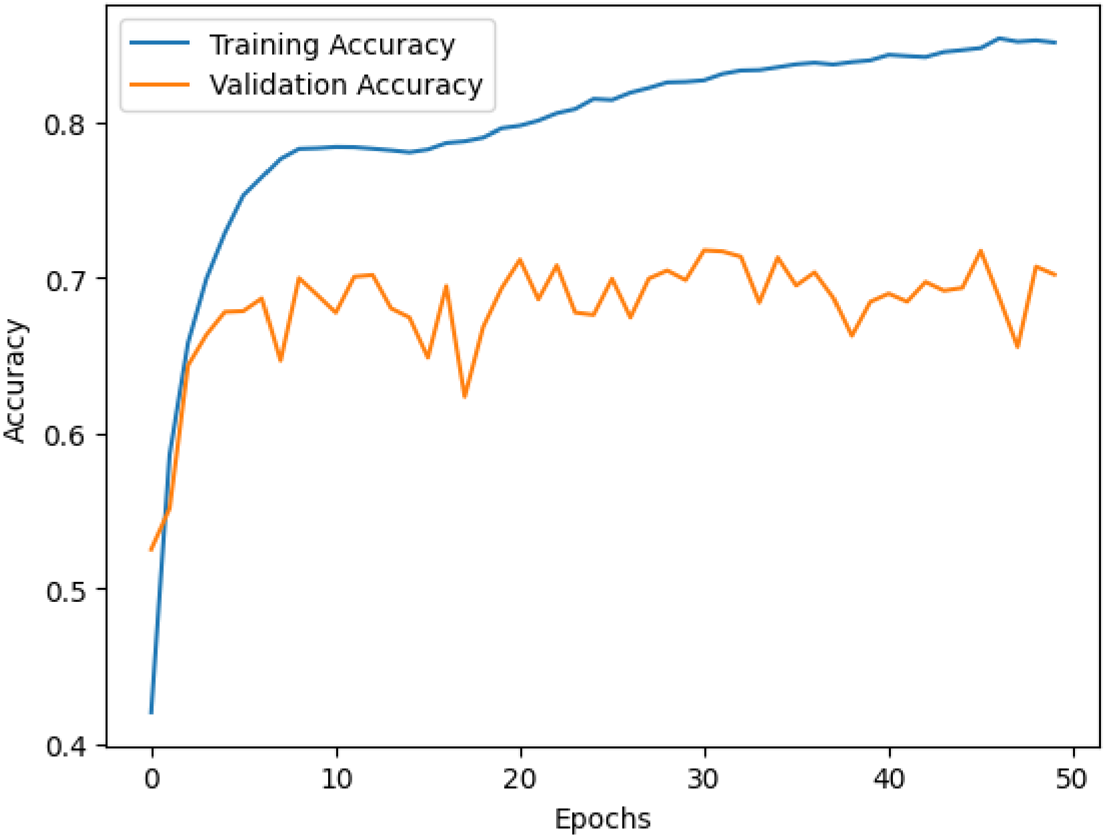
```
model = models.Sequential()

# Add first convolutional layer
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))

# Add second convolutional layer
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Add third convolutional layer
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Flattening the results
model.add(layers.Flatten())

# Add first dense layer (hidden units increased to 256)
model.add(layers.Dense(256, activation='relu'))

# Add second dense layer to increase hidden layers
model.add(layers.Dense(128, activation='relu'))

# Add output layer
model.add(layers.Dense(10, activation='softmax'))

cnnModel.summary()
```

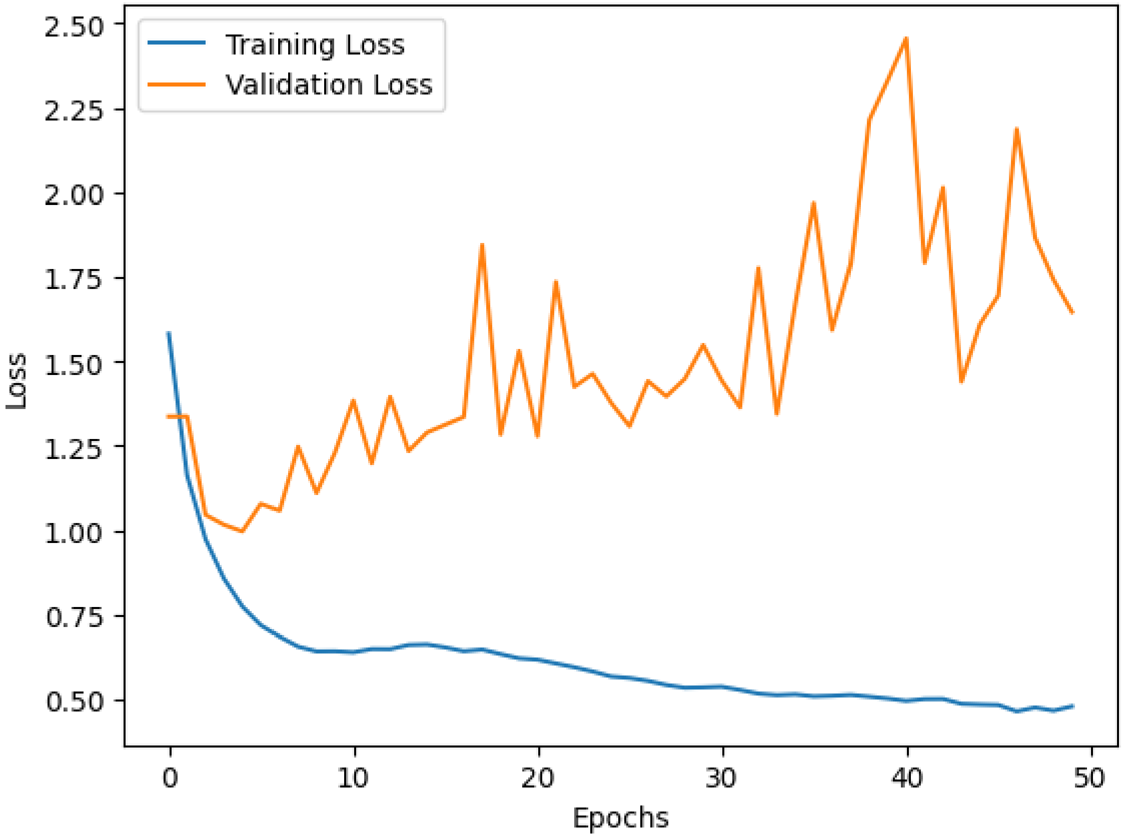
Model: "sequential"

```
┃ Layer (type)                      ┃ Output Shape              ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
│ conv2d (Conv2D)                   │ (None, 26, 26, 32)        │
├───────────────────────────────────┼───────────────────────────┼
│ max_pooling2d (MaxPooling2D)      │ (None, 13, 13, 32)        │
├───────────────────────────────────┼───────────────────────────┼
│ conv2d_1 (Conv2D)                 │ (None, 11, 11, 64)        │
├───────────────────────────────────┼───────────────────────────┼
│ max_pooling2d_1 (MaxPooling2D)    │ (None, 5, 5, 64)          │
├───────────────────────────────────┼───────────────────────────┼
│ conv2d_2 (Conv2D)                 │ (None, 3, 3, 64)          │
├───────────────────────────────────┼───────────────────────────┼
│ flatten (Flatten)                 │ (None, 576)               │
├───────────────────────────────────┼───────────────────────────┼
│ dense (Dense)                     │ (None, 64)                │
├───────────────────────────────────┼───────────────────────────┼
│ dense_1 (Dense)                   │ (None, 32)                │
├───────────────────────────────────┼───────────────────────────┼
│ dense_2 (Dense)                   │ (None, 10)                │
└───────────────────────────────────┴───────────────────────────┴

 Total params: 285,248 (1.09 MB)
 Trainable params: 95,082 (371.41 KB)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 190,166 (742.84 KB)
```

In [21]:

```python
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plotting training and validation loss over epochs
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Evaluate the model on the test dataset
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"Test accuracy: {test_acc}")

# Save the model if needed
model.save('cnn_cifar10_model.h5')
```





```
313/313 - 5s - 18ms/step - accuracy: 0.7022 - loss: 1.6467
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.sa
ving.save_model(model)`. This file format is considered legacy. We recommend using ins
tead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.sav
e_model(model, 'my_model.keras')`.
Test accuracy: 0.7021999955177307
```