CONVOLUTIONAL NEURAL NETWORK

PROJECT REPORT
AND
DOCUMENTATION

SUBMITTED BY,

CHINMAYA A S

Forest Fire Detection Using Convolutional Neural Network (CNN)

Introduction

Forest fires cause severe environmental and economic damage. Early detection is crucial for minimizing loss. This project implements a **Convolutional Neural Network (CNN)** to automatically classify images into **Fire** and **No Fire** categories using deep learning techniques.

The system takes an image as input, processes it through multiple convolution and pooling layers to extract meaningful visual features, and finally predicts whether fire is present in the image.

Objective of the Project

- To build an image classification model using CNN
- To detect forest fires from images
- To understand end-to-end CNN workflow: data loading → preprocessing → training → evaluation
- To visualize model performance using a confusion matrix

Technologies and Libraries Used

Technology	Purpose
Python	Programming language
OpenCV (cv2)	Image reading and resizing
NumPy	Numerical operations
Scikit-learn	Label encoding, train-test split, evaluation
TensorFlow / Keras	CNN model building and training
Seaborn	Confusion matrix visualization

Dataset Description

The dataset consists of images divided into two classes:

- **fire** – images containing forest fire
- **nofire** – images without fire

Folder Structure

forest_fire/

├── Training and Validation/

│ ├── fire/

│ └── nofire/

└── Testing/

├── fire/

└── nofire/

Each folder name represents the class label.

Code Documentation and Explanation

1.Warning Suppression

```
import warnings
warnings.filterwarnings("ignore")
```

Purpose:

- Suppresses non-critical warning messages from libraries
- Keeps output clean and readable

---

2. Importing Required Libraries

```
import os
import cv2
import numpy as np
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from tensorflow.keras.models import Sequential,load_model
from tensorflow.keras.layers import Conv2D,MaxPool2D,Flatten,Dense
```

Explanation:

- os: Used to navigate folders and read directory contents
- cv2: Reads and resizes images
- numpy: Handles numerical data and arrays
- seaborn: Used for plotting confusion matrix
- LabelEncoder: Converts class labels from text to numbers
- train_test_split: Splits data into training and testing sets
- confusion_matrix: Evaluates classification performance
- Sequential: Builds CNN layer by layer
- CNN layers: Used to extract features and classify images

3.Loading and Preprocessing Images (Testing Data)

```
data=r"C:\Users\chinm\Downloads\forest_fire-20260130T082050Z-3-001\forest_fire\Testing"
```

```
x1,y1=[],[]
for i in os.listdir(data):
    new_dir=os.path.join(data,i)
    if os.path.isdir(new_dir):
        for j in os.listdir(new_dir):
            image_path=os.path.join(new_dir,j)
            if image_path.endswith((".png",".jpg",".jpeg")):
                image_data=cv2.imread(image_path)
                array_resize=cv2.resize(image_data,(100,100))
                array_reshape=array_resize.reshape(100,100,3)
                array_norm=array_reshape/255
                x1.append(array_norm)
                y1.append(i)
x1=np.array(x1)
y1=np.array(y1)
print(x1.shape)
print(y1.shape)
```

```
(68, 100, 100, 3)
(68,)
```

- x1: Stores image pixel data
- y1: Stores corresponding labels
- Iterates through class folders (fire, nofire)
- Folder name acts as the label
- Reads image and converts it to a NumPy array
- Resizes all images to 100×100 pixels for uniform input size
- Normalizes pixel values from range **0–255** to **0–1**
- Improves model convergence and numerical stability
- Stores processed image and its label

The same preprocessing steps are applied to training and validation data to maintain consistency.

```
data1=r"E:\CNN_DATASET\forest_fire-20260130T082050Z-3-001\forest_fire\Training and Validation"
```

```
x2,y2=[],[]
for i in os.listdir(data1):
    new_dir=os.path.join(data1,i)
    if os.path.isdir(new_dir):
        for j in os.listdir(new_dir):
            image_path=os.path.join(new_dir,j)
            if image_path.endswith((".png",".jpg",".jpeg")):
                image_data=cv2.imread(image_path)
                array_resize=cv2.resize(image_data,(100,100))
                array_reshape=array_resize.reshape(100,100,3)
                array_norm=array_reshape/255
                x2.append(array_norm)
                y2.append(i)
x2=np.array(x2)
y2=np.array(y2)
print(x2.shape)
print(y2.shape)
```

```
(1912, 100, 100, 3)
(1912,)
```

4.Combining Datasets

```
x=np.concatenate((x1,x2))
y=np.concatenate((y1,y2))
y
```

```
array(['fire', 'fire', 'fire', ..., 'nofire', 'nofire', 'nofire'],
      dtype='<U6')
```

Why this is done:

- Allows a fresh and randomized train-test split
- Ensures balanced data distribution

5.Label Encoding

```
le=LabelEncoder()
y=le.fit_transform(y)
y
```

```
array([0, 0, 0, ..., 1, 1, 1])
```

Explanation:

- Converts string labels into numeric form

- Example:
    - fire → 0
    - nofire → 1
- Required because neural networks work with numbers, not text

---

6.Train-Test Split

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state=42)
```

```
print(len(xtrain),len(ytrain))
print(len(xtest),len(ytest))
```

Explanation:

- 80% data for training
- 20% data for testing
- random_state=42 ensures reproducibility

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 98, 98, 32)	896
max_pooling2d (MaxPooling2D)	(None, 49, 49, 32)	0
conv2d_1 (Conv2D)	(None, 47, 47, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 23, 23, 64)	0
conv2d_2 (Conv2D)	(None, 21, 21, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 10, 10, 128)	0
flatten (Flatten)	(None, 12800)	0
dense (Dense)	(None, 128)	1,638,528
dense_1 (Dense)	(None, 2)	258

Total params: 1,732,034 (6.61 MB)

Trainable params: 1,732,034 (6.61 MB)

Non-trainable params: 0 (0.00 B)

7.CNN Model Architecture

```
model=Sequential()
model.add(Conv2D(32,(3,3),activation="relu",input_shape=(100,100,3)))
model.add(MaxPool2D(2,2))
model.add(Conv2D(64,(3,3),activation="relu"))
model.add(MaxPool2D(2,2))
model.add(Conv2D(128,(3,3),activation="relu"))
model.add(MaxPool2D(2,2))
model.add(Flatten())
model.add(Dense(128,activation="relu"))
model.add(Dense(2,activation="softmax"))
```

- Initializes a sequential CNN model

7.1 Convolution Layer

- Extracts low-level features such as edges and textures
- 32 filters with size 3×3

7.2 Max Pooling Layer

- Reduces spatial dimensions
- Retains important features

7.3 Deeper Convolution Layers

- Learns complex patterns like fire shapes and smoke regions

7.4 Flatten Layer

- Converts feature maps into a 1D vector

7.5 Fully Connected Layers

- Final classification layers
- Softmax outputs probability for each class

---

8. Model Compilation

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

Explanation:

- **Adam optimizer**: Fast and adaptive learning
- **Sparse categorical crossentropy**: Used because labels are integers
- **Accuracy**: Performance metric

---

9. Model Training

```
model.fit(xtrain,ytrain,batch_size=10,epochs=22,validation_data=(xtest,ytest))
```

```
Epoch 1/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 49s 280ms/step - accuracy: 0.8883 - loss: 0.3021 - val_accuracy: 0.9268 - val_loss: 0.1708
Epoch 2/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 82s 282ms/step - accuracy: 0.9451 - loss: 0.1726 - val_accuracy: 0.9091 - val_loss: 0.2444
Epoch 3/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 33s 208ms/step - accuracy: 0.9369 - loss: 0.1893 - val_accuracy: 0.9520 - val_loss: 0.1184
Epoch 4/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 45s 226ms/step - accuracy: 0.9514 - loss: 0.1355 - val_accuracy: 0.9697 - val_loss: 0.1191
Epoch 5/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 33s 205ms/step - accuracy: 0.9646 - loss: 0.1127 - val_accuracy: 0.9621 - val_loss: 0.0876
Epoch 6/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 44s 222ms/step - accuracy: 0.9621 - loss: 0.1147 - val_accuracy: 0.9697 - val_loss: 0.0926
Epoch 7/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 33s 209ms/step - accuracy: 0.9691 - loss: 0.0836 - val_accuracy: 0.9596 - val_loss: 0.1146
Epoch 8/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 49s 258ms/step - accuracy: 0.9823 - loss: 0.0566 - val_accuracy: 0.9545 - val_loss: 0.1227
Epoch 9/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 41s 254ms/step - accuracy: 0.9817 - loss: 0.0522 - val_accuracy: 0.9848 - val_loss: 0.0684
Epoch 10/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 43s 271ms/step - accuracy: 0.9905 - loss: 0.0293 - val_accuracy: 0.9798 - val_loss: 0.0959

Epoch 11/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 87s 303ms/step - accuracy: 0.9848 - loss: 0.0433 - val_accuracy: 0.9848 - val_loss: 0.0718
Epoch 12/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 62s 387ms/step - accuracy: 0.9811 - loss: 0.0684 - val_accuracy: 0.9823 - val_loss: 0.0737
Epoch 13/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 57s 230ms/step - accuracy: 0.9962 - loss: 0.0149 - val_accuracy: 0.9747 - val_loss: 0.1272
Epoch 14/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 36s 228ms/step - accuracy: 0.9956 - loss: 0.0138 - val_accuracy: 0.9646 - val_loss: 0.1794
Epoch 15/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 33s 208ms/step - accuracy: 0.9962 - loss: 0.0097 - val_accuracy: 0.9747 - val_loss: 0.1325
Epoch 16/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 34s 210ms/step - accuracy: 0.9987 - loss: 0.0053 - val_accuracy: 0.9823 - val_loss: 0.1177
Epoch 17/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 49s 258ms/step - accuracy: 1.0000 - loss: 6.0002e-04 - val_accuracy: 0.9798 - val_loss: 0.1384
Epoch 18/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 77s 226ms/step - accuracy: 1.0000 - loss: 2.1154e-04 - val_accuracy: 0.9773 - val_loss: 0.1448
Epoch 19/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 43s 238ms/step - accuracy: 1.0000 - loss: 1.2758e-04 - val_accuracy: 0.9773 - val_loss: 0.1479
Epoch 20/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 41s 237ms/step - accuracy: 1.0000 - loss: 9.3052e-05 - val_accuracy: 0.9773 - val_loss: 0.1536
Epoch 21/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 35s 200ms/step - accuracy: 1.0000 - loss: 7.3144e-05 - val_accuracy: 0.9773 - val_loss: 0.1603
Epoch 22/22
159/159 ━━━━━━━━━━━━━━━━━━━━ 41s 259ms/step - accuracy: 1.0000 - loss: 5.5550e-05 - val_accuracy: 0.9773 - val_loss: 0.1625
<keras.src.callbacks.history.History at 0x23c6046f380>
```

- Batch size: 10 images per iteration
- Epochs: Model sees full dataset 22 times
- Validation data monitors overfitting

10.Image Loading and Preprocessing for CNN Prediction

```
test_data1=r"E:\CNN_DATASET\forest_fire-20260130T082050Z-3-001\forest_fire\Training and Validation\fire\abc005.jpg"
```

```
image_data=cv2.imread(test_data1)
array_resize=cv2.resize(image_data,(100,100))
array_reshape=array_resize.reshape(1,100,100,3)
array_norm=array_reshape/255
print(array_norm.shape)
```

```
(1, 100, 100, 3)
```

- The image file path is stored as a raw string so that Windows backslashes are interpreted correctly and the file can be accessed without errors.
- The image is loaded from disk into memory using OpenCV and converted into a NumPy array containing pixel values arranged as height, width, and color channels (BGR format).
- The image is resized to a fixed dimension of 100 × 100 pixels to match the input size expected by the CNN, ensuring consistency with the data used during training.
- A batch dimension is added to the image data so that it follows the CNN input format (batch_size, height, width, channels), even though only a single image is being processed.
- Pixel values are normalized from the range 0–255 to 0–1, which improves numerical stability and ensures the model behaves the same way during prediction as it did during training.
- The final shape of the processed image is checked to confirm that it is compatible with the trained model and ready for prediction.

11. Model Prediction and Class Interpretation

```
a=model.predict(array_norm)
a
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 488ms/step
array([[1.000000e+00, 4.016012e-15]], dtype=float32)
```

```
label=["fire","nofire"]
```

```
b=np.argmax(a)  #argmax give index of largest value
label[b]
```

```
'fire'
```

- The preprocessed image is passed into the trained CNN model, which performs a forward pass through all convolutional, pooling, and dense layers to generate prediction values.
- The model outputs an array of **probability scores**, where each value represents how confident the model is about a particular class.
- A label list is defined to map numerical class indices to meaningful class names, ensuring the prediction output can be easily understood by humans.
- The class with the highest probability is selected by finding the index of the maximum value in the prediction output.

- This index corresponds to the predicted class, which is then converted into its actual label name (either *fire* or *nofire*).
- The final result represents the model's decision for the given image based on learned patterns during training.

---

12. No-Fire Image Prediction and Result Interpretation

```
test_data2=r"C:\Users\chinm\Downloads\forest_fire-20260130T082050Z-3-001\forest_fire\Testing\nofire\abc336.jpg"
```

```
image_data=cv2.imread(test_data2)
array_resize=cv2.resize(image_data,(100,100))
array_reshape=array_resize.reshape(1,100,100,3)
array_norm=array_reshape/255
print(array_norm.shape)
```

```
(1, 100, 100, 3)
```

```
c=model.predict(array_norm)
c
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 135ms/step
array([[2.4931949e-06, 9.9999750e-01]], dtype=float32)
```

```
d=np.argmax(c)
label[d]
```

```
'nofire'
```

---

13. Batch Prediction and Class Label Extraction

```
ypred=model.predict(xtest)
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 2s 140ms/step
```

```
ypredd=[]
for i in ypred:
    ypredd.append(np.argmax(i))
```

- The trained model is used to generate predictions for the entire test dataset at once, producing probability scores for each test image.
- For every test image, the model outputs a set of values representing the confidence for each possible class.
- These probability values cannot be directly compared with actual class labels, so they must be converted into discrete class predictions.
- For each prediction, the class with the highest probability is selected as the model's final decision.

- The selected class indices are stored in a new list, creating a complete set of predicted labels for the test dataset.
- This processed prediction list is now suitable for performance evaluation using metrics such as accuracy and confusion matrix.

---

14. Confusion Matrix Creation and Interpretation

```
cm=confusion_matrix(ytest,ypredd)
cm
```

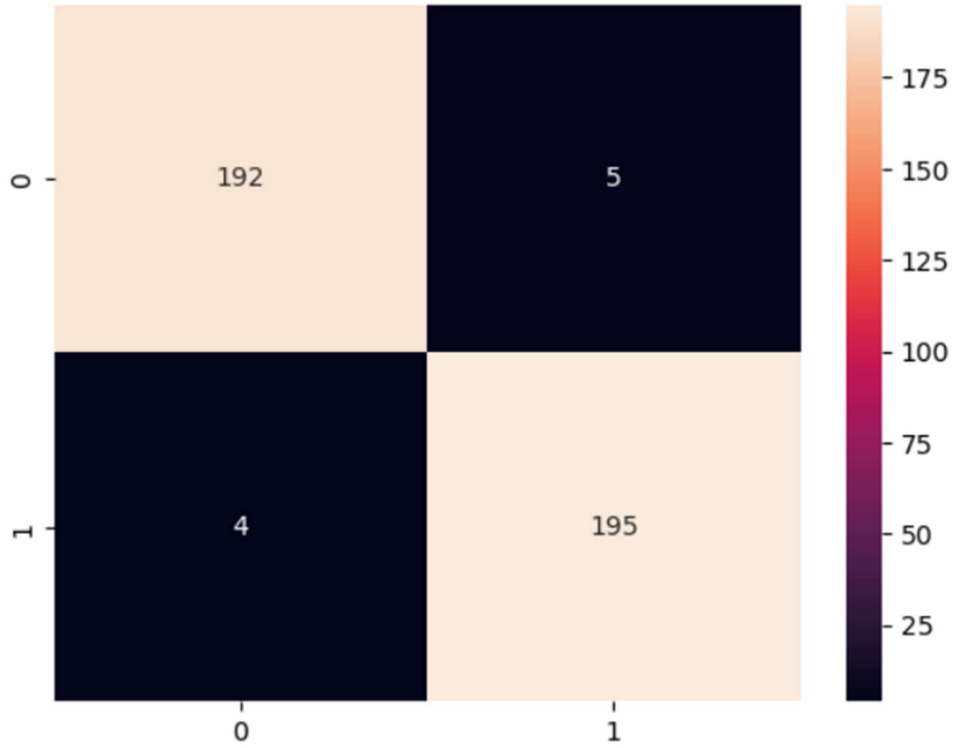
```
array([[192,   5],
       [  4, 195]])
```

- The confusion matrix is generated by comparing the **actual class labels** from the test dataset with the **predicted class labels** produced by the model.
- Internally, the function counts how many predictions fall into each category of correct and incorrect classification.
- Each row of the matrix represents the **true class**, while each column represents the **predicted class**.
- The diagonal values indicate **correct predictions**, where the model correctly identified the class.
- The off-diagonal values represent **misclassifications**, showing where the model confused one class with another.
- This matrix provides a detailed view of model performance beyond accuracy, helping identify false positives and false negatives.
- The confusion matrix output is commonly used for further evaluation metrics and visualization.

15. Confusion Matrix Visualization Using Heatmap

```
sns.heatmap(cm,fmt="d",annot=True)
```

```
<Axes: >
```



- The confusion matrix values are converted into a visual representation to make model performance easier to interpret.
- Each cell in the heatmap represents the number of predictions for a specific actual vs predicted class combination.
- Numerical values are displayed inside each cell, allowing exact counts to be read directly from the visualization.
- Darker or more intense colors indicate higher values, making correct and incorrect predictions visually distinct.

16.Results and Conclusion

- The CNN successfully classifies forest fire images
- Convolution layers effectively extract visual patterns
- Pooling layers reduce computation while preserving important features
- The model can be used as a foundation for real-time fire detection systems