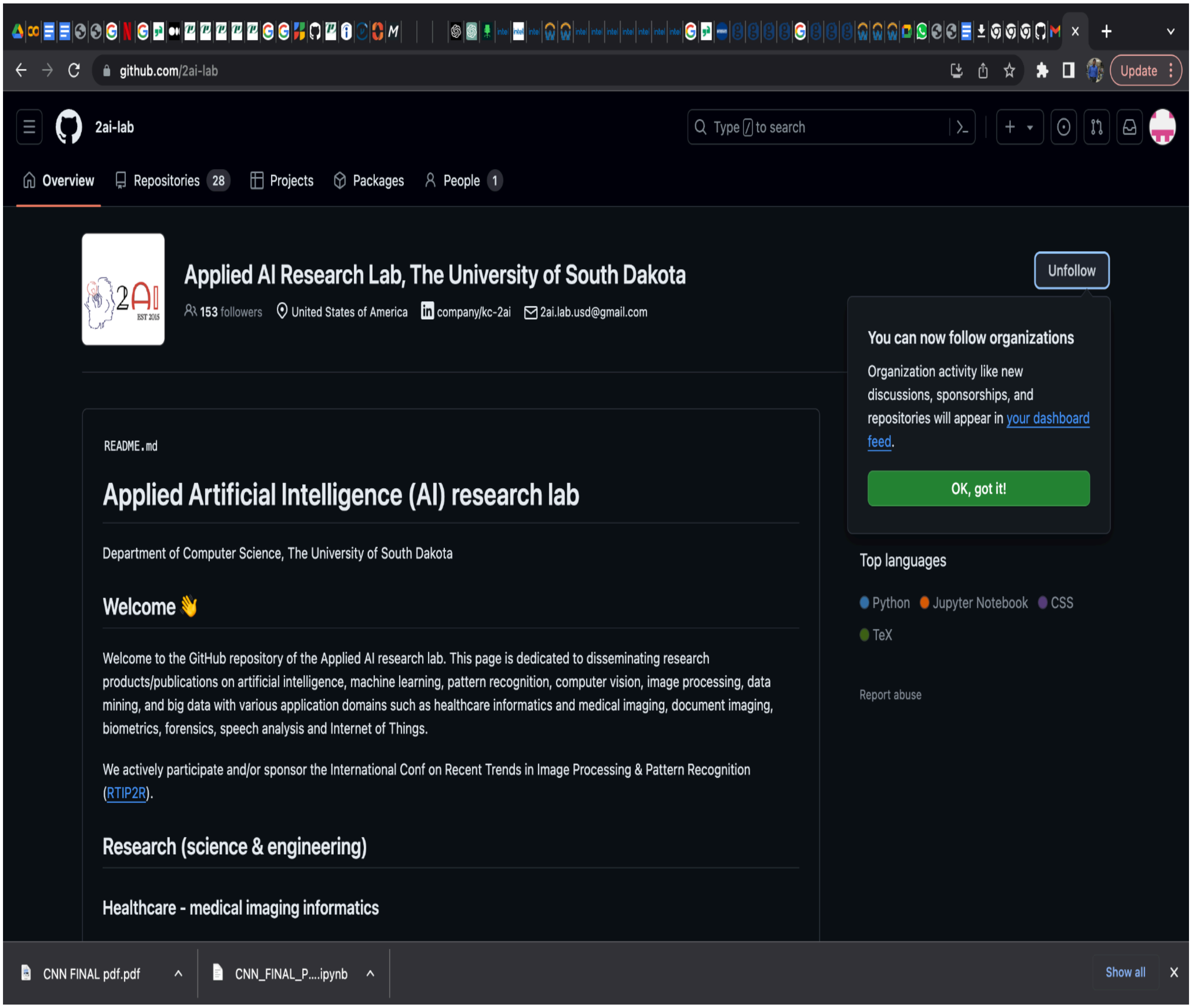
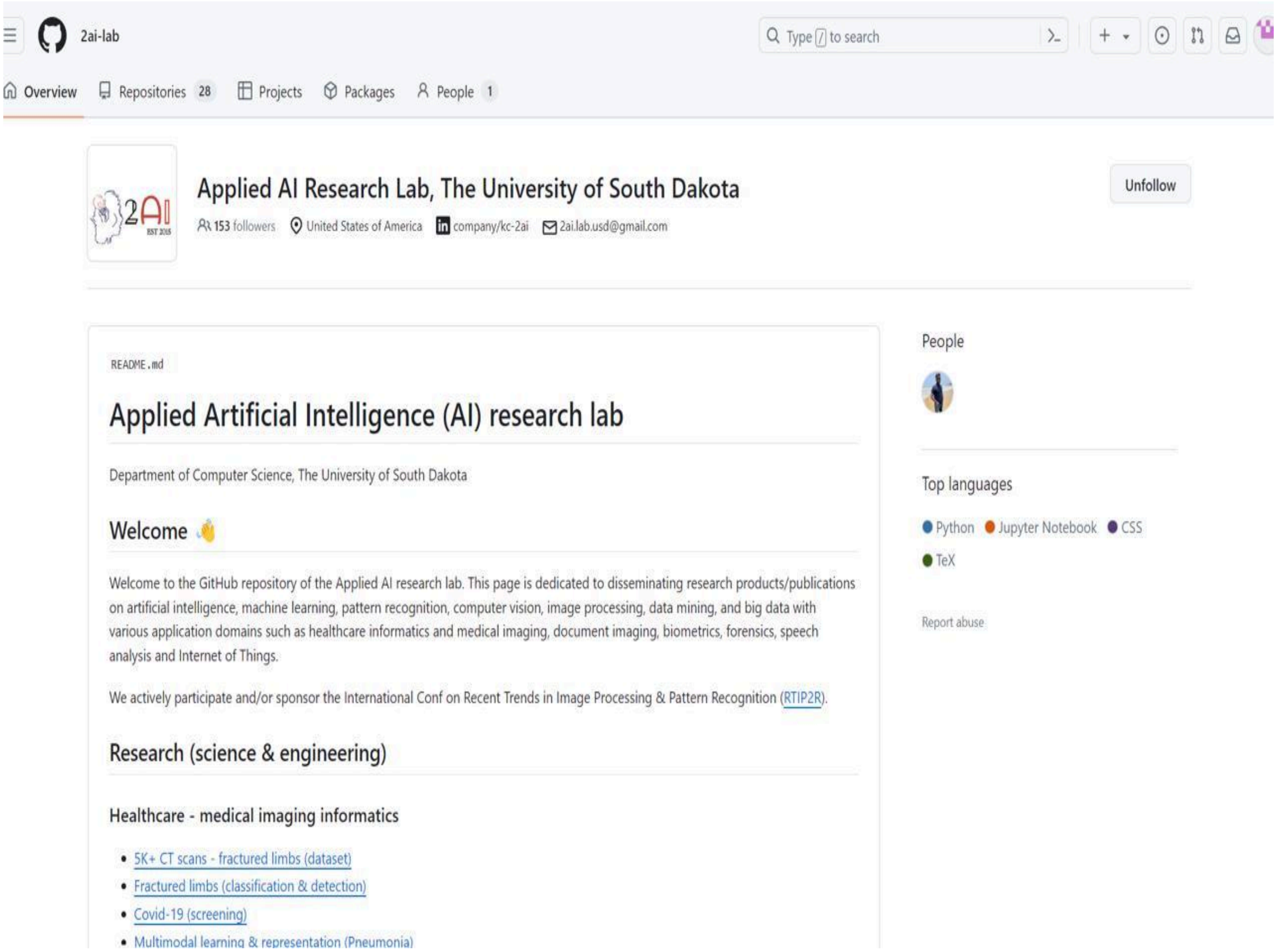


VENKATA SAI BHARGAV VEERAMSETTY



2ai-lab

Type / to search

Overview Repositories 28 Projects Packages People 1

Applied AI Research Lab, The University of South Dakota

153 followers · United States of America · company/kc-2ai · 2ai.lab.usd@gmail.com

Unfollow

README.md

Applied Artificial Intelligence (AI) research lab

Department of Computer Science, The University of South Dakota

Welcome

Welcome to the GitHub repository of the Applied AI research lab. This page is dedicated to disseminating research products/publications on artificial intelligence, machine learning, pattern recognition, computer vision, image processing, data mining, and big data with various application domains such as healthcare informatics and medical imaging, document imaging, biometrics, forensics, speech analysis and Internet of Things.

We actively participate and/or sponsor the International Conf on Recent Trends in Image Processing & Pattern Recognition (RTIP2R).

Research (science & engineering)

Healthcare - medical imaging informatics

- 5K+ CT scans - fractured limbs (dataset)
- Fractured limbs (classification & detection)
- Covid-19 (screening)
- Multimodal learning & representation (Pneumonia)

People

Top languages

Python · Jupyter Notebook · CSS · TeX

Report abuse

SIVA PRASAD REDDY NALAMARU

```
CNN final project


Team Members:
Name ID
Venkata Sai Bhargav VEERAMSETTY - 101149826
Siva prasad reddy nalamaru - 101139281
Kareem Abdul-  101111784
Mahabub Shaariief Shaik - 101163600
Hussainamma Shaik - 101159835
Sai Harshitha Laggu -  101163328
Yesaswini Kalahasti - 101165876
```

CNN final project

Team Members: Name ID Venkata Sai Bhargav VEERAMSETTY - 101149826 Siva prasad reddy nalamaru - 101139281 Kareem Abdul- 101111784 Mahabub Shaariief Shaik - 101163600 Hussainamma Shaik - 101159835 Sai Harshitha Laggu - 101163328 Yesaswini Kalahasti - 101165876

With the MNIST dataset, a Convolutional Neural Network (CNN) for handwritten digit recognition is implemented in this research. We first preprocess the data before designing a CNN architecture that consists of fully connected, max-pooling, and convolutional layers. The model is trained on the dataset, and its performance is assessed through the use of measures like accuracy and loss curves. For a robust examination, K-Fold Cross Validation is used. Along with schematics showing the CNN architecture, the project offers comprehensive documentation of the specifications and measurements of each network component. Results analysis provides information about how CNN components affect performance as well as how to interpret the confusion matrix. This project provides hands-on experience with CNN implementation for model evaluation and image categorization.

Importing Libariers

```
pip install ucimlrepo
```

```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.6-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.6
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import confusion_matrix, accuracy_score
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation
from keras.utils import to_categorical
from ucimlrepo import fetch_ucirepo
```

1) Data Preparation & Importing Data Sets

```
from sklearn.datasets import load_digits

# fetch dataset
digits = load_digits()

# data
X = digits.data
y = digits.target

# metadata
print(digits.DESCR)

# feature names
print(digits.feature_names)
```

```
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**
```

```
    :Number of Instances: 1797
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
    Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
    Linear dimensionalityreduction using relevance weighted LDA. School of
    Electrical and Electronic Engineering Nanyang Technological University.
    2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification
    Algorithm. NIPS. 2000.

['pixel_0_0', 'pixel_0_1', 'pixel_0_2', 'pixel_0_3', 'pixel_0_4', 'pixel_0_5', 'pixel_0_6', 'pixel_0_7', 'pixel_1_0', 'pixel_1_1', 'pixe
```

Preprocesing started..

```
from sklearn.datasets import load_digits

# Load the dataset
digits = load_digits()

# Extracting the feature and target variables
X = digits.data
y = digits.target
```

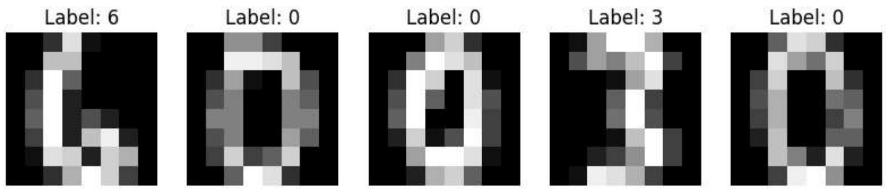
```
from sklearn.model_selection import train_test_split
import numpy as np

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Preprocess the data (Normalization and Reshaping)
X_train = X_train.astype('float32') / 16.0  # Normalization
X_test = X_test.astype('float32') / 16.0    # Normalization
X_train = np.expand_dims(X_train, axis=-1)  # Reshaping
X_test = np.expand_dims(X_test, axis=-1)    # Reshaping
```

```
import pandas as pd

# Reshape data to 2D
X_train_reshaped = X_train.reshape(X_train.shape[0], -1)  # Reshaping to (number of samples, 64)
X_test_reshaped = X_test.reshape(X_test.shape[0], -1)  # Reshaping to (number of samples, 64)

# Create pandas DataFrames
train_df = pd.DataFrame(X_train_reshaped, columns=[f"Pixel_{i}" for i in range(X_train_reshaped.shape[1])])
test_df = pd.DataFrame(X_test_reshaped, columns=[f"Pixel_{i}" for i in range(X_test_reshaped.shape[1])])

# Display the first few rows of each DataFrame
print("Training Data:")
print(train_df.head())
print("\nTesting Data:")
print(test_df.head())
```

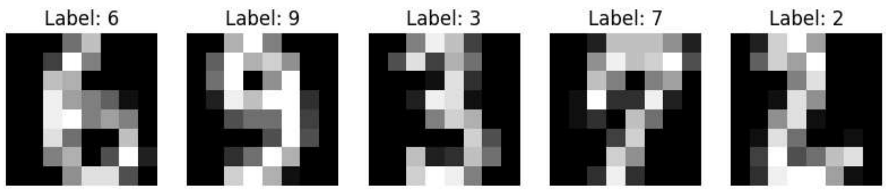
```
Training Data:
   Pixel_0  Pixel_1  Pixel_2  Pixel_3  Pixel_4  Pixel_5  Pixel_6  Pixel_7  \
0      0.0   0.0000   0.1875   0.8750   0.0625   0.0000      0.0      0.0
1      0.0   0.0000   0.5625   0.5625   0.2500   0.0000      0.0      0.0
2      0.0   0.0000   0.0000   0.6250   0.8125   0.1875      0.0      0.0
3      0.0   0.0625   0.6250   1.0000   1.0000   0.6875      0.0      0.0
4      0.0   0.0000   0.3750   0.8750   0.8125   0.1875      0.0      0.0

   Pixel_8  Pixel_9  ...  Pixel_54  Pixel_55  Pixel_56  Pixel_57  Pixel_58  \
0      0.0   0.0000  ...    0.6875       0.0       0.0    0.0000    0.1875
1      0.0   0.0000  ...    0.0000       0.0       0.0    0.0000    0.3750
2      0.0   0.0000  ...    0.0625       0.0       0.0    0.0000    0.1250
3      0.0   0.3125  ...    0.2500       0.0       0.0    0.0625    0.9375
4      0.0   0.0000  ...    0.1250       0.0       0.0    0.0000    0.2500

   Pixel_59  Pixel_60  Pixel_61  Pixel_62  Pixel_63
0    0.6875    1.0000    0.8125      0.25       0.0
1    1.0000    0.8750    0.1875      0.00       0.0
2    0.6875    0.8125    0.3750      0.00       0.0
3    0.8750    0.6875    0.2500      0.00       0.0
4    0.9375    1.0000    0.5625      0.00       0.0

[5 rows x 64 columns]

Testing Data:
   Pixel_0  Pixel_1  Pixel_2  Pixel_3  Pixel_4  Pixel_5  Pixel_6  Pixel_7  \
0      0.0    0.000   0.0000   0.4375    0.750     0.00   0.0000    0.000
1      0.0    0.000   0.6875   1.0000    0.500     0.00   0.0000    0.000
2      0.0    0.000   0.5000   0.9375    0.750     0.25   0.0000    0.000
3      0.0    0.000   0.1250   0.7500    0.750     0.75   0.5625    0.125
4      0.0    0.125   0.8125   1.0000    0.625     0.00   0.0000    0.000

   Pixel_8  Pixel_9  ...  Pixel_54  Pixel_55  Pixel_56  Pixel_57  Pixel_58  \
0      0.0   0.0000  ...    1.0000     0.125       0.0    0.0000    0.0000
1      0.0   0.3750  ...    0.0000     0.000       0.0    0.0000    0.8125
2      0.0   0.3125  ...    0.4375     0.000       0.0    0.0000    0.8125
3      0.0   0.0000  ...    0.0000     0.000       0.0    0.0000    0.1875
4      0.0   0.3750  ...    0.8750     0.000       0.0    0.1875    0.9375

   Pixel_59  Pixel_60  Pixel_61  Pixel_62  Pixel_63
0    0.5625    0.8750    0.8750    0.3125       0.0
1    1.0000    0.6875    0.0625    0.0000       0.0
2    1.0000    0.9375    0.5000    0.0000       0.0
3    0.9375    0.1875    0.0000    0.0000       0.0
4    1.0000    1.0000    0.6250    0.0625       0.0

[5 rows x 64 columns]
```

```
import matplotlib.pyplot as plt

# Function to display images
def display_images(images, labels, num_images=5):
    fig, axes = plt.subplots(1, num_images, figsize=(10, 3))
    for i in range(num_images):
        axes[i].imshow(images[i].reshape(8, 8), cmap='gray')
        axes[i].set_title(f'Label: {labels[i]}')
        axes[i].axis('off')
    plt.show()

# Display the first few images and labels from the training and testing datasets
print("First few training images and their labels:")
display_images(X_train, y_train)

print("First few testing images and their labels:")
display_images(X_test, y_test)
```

```
First few training images and their labels:
```




```
First few testing images and their labels:
```



In order to guarantee stable training, avoid optimization problems, equalize feature priority, regularize the model, and enhance interpretability, normalization is necessary. Normalization improves the model's performance.

Convolutiomnal Neural Network Architecture

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D

# Define CNN architecture
model = Sequential()

# Convolutional layers
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(8, 8, 1), name="Conv2D_1"))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', name="Conv2D_2"))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', name="Conv2D_3"))
model.add(Conv2D(256, kernel_size=(2, 2), activation='relu', name="Conv2D_4"))
```

1) Four convolutional layers make up the architecture, and a ReLU activation function comes after each.

2) 32 filters of size (3, 3) are applied to input pictures with the shape (8, 8, 1) by the first convolutional layer (Conv2D_1).

3) The number of filters in subsequent layers is increased, which may enable the capture of more intricate patterns in the data.

4) With a smaller kernel size of (2, 2), the final convolutional layer (Conv2D_4) may be able to capture more minute details in the feature maps.

Max Pooling

```
model.pop()

# Add the max pooling layer again with the same name
model.add(MaxPooling2D((2, 2), name="MaxPooling2D_1"))
```

The MaxPooling2D layers use the maximum value in each region determined by the pool size to downsample feature maps.

minimizes spatial dimensions while keeping significant characteristics.

Flattern Layer

```
#flattern layer
model.add(Flatten(name="Flatten"))
```

4)Fully Connected Layer and Softmax

```
from tensorflow.keras.layers import Dense

# Add Dense layers
model.add(Dense(64, activation='relu', name="Dense_1"))
model.add(Dense(10, activation='softmax', name="Output"))
```

Obserrvation : Convolutional layers' output is transformed into a 1D array by the flatten layer.

The 64 neurons in the Dense_1 layer are activated by ReLU. Ten neurons in the output layer are activated using softmax for categorization. These convolutional layers extract features, which these dense layers use to accomplish classification. Every neuron in the dense layer is linked to every other neuron in the layer above it. To add non-linearity, ReLU activation functions are frequently employed in hidden dense layers. For multi-class classification tasks, where each neuron reflects the likelihood of a specific class, the output layer usually uses softmax activation. As there are ten classes in this classification task, there are ten neurons in the output layer.Convolutional layers' output is transformed into a 1D array by the flatten layer. 64 neurons in the Dense_1 layer are ReLU activated. Ten neurons in the output layer have softmax activity for categorization.

Based on the features that the convolutional layers extracted, these dense layers classify data. Every neuron in the layer above is linked to every other neuron in the dense layer. ReLU activation functions are frequently employed to add non-linearity to hidden dense layers. For multi-class classification problems, the output layer usually uses softmax activation, where each neuron represents the likelihood of a specific class. Ten neurons make up the output layer in this instance, which is equivalent to the ten classes in the classification test.

parameters and dimensions.

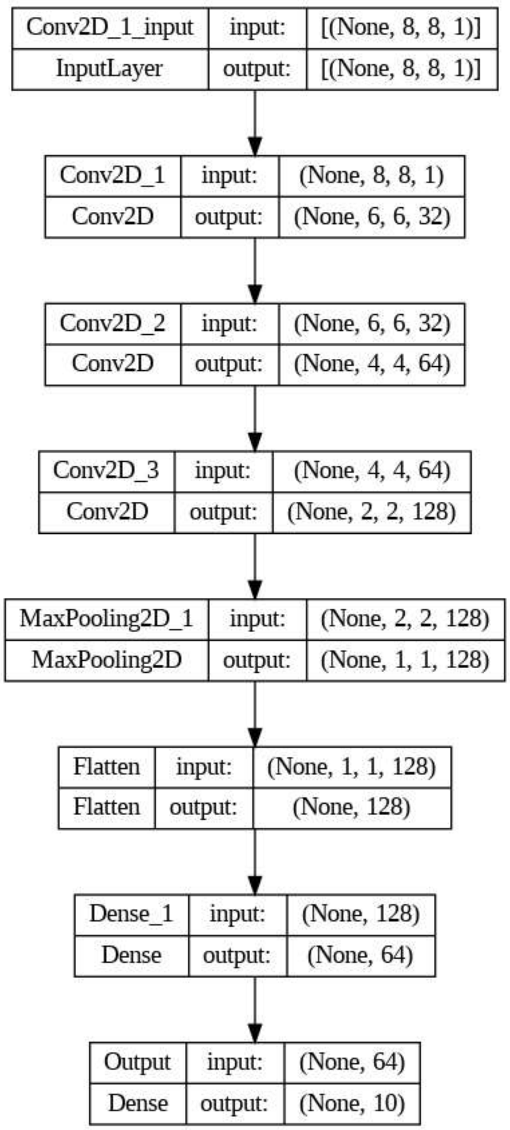
```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
#model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Conv2D_1 (Conv2D)           (None, 6, 6, 32)          320

 Conv2D_2 (Conv2D)           (None, 4, 4, 64)          18496

 Conv2D_3 (Conv2D)           (None, 2, 2, 128)         73856

 MaxPooling2D_1 (MaxPooling  (None, 1, 1, 128)         0
 2D)

 Flatten (Flatten)           (None, 128)               0

 Dense_1 (Dense)             (None, 64)                8256

 Output (Dense)              (None, 10)                650

=================================================================
Total params: 101578 (396.79 KB)
Trainable params: 101578 (396.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
pip install pydot graphviz
```

```
Requirement already satisfied: pydot in /usr/local/lib/python3.10/dist-packages (1.4.2)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (0.20.3)
Requirement already satisfied: pyparsing>=2.1.4 in /usr/local/lib/python3.10/dist-packages (from pydot) (3.1.2)
```

```
from tensorflow.keras.utils import plot_model

# Assuming your model is stored in the variable 'model'
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```



5) Training Process

```
model = Sequential()
model.add(Flatten(input_shape=(64, 1)))
# ... add other layers ...
```

```
from tensorflow.keras.utils import plot_model

# Assuming your model is stored in the variable 'model'
model.compile(optimizer='adam', loss='mean_squared_error')
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)

history = model.fit(X_train, y_train, epochs=50, batch_size=64, validation_split=0.2)
```

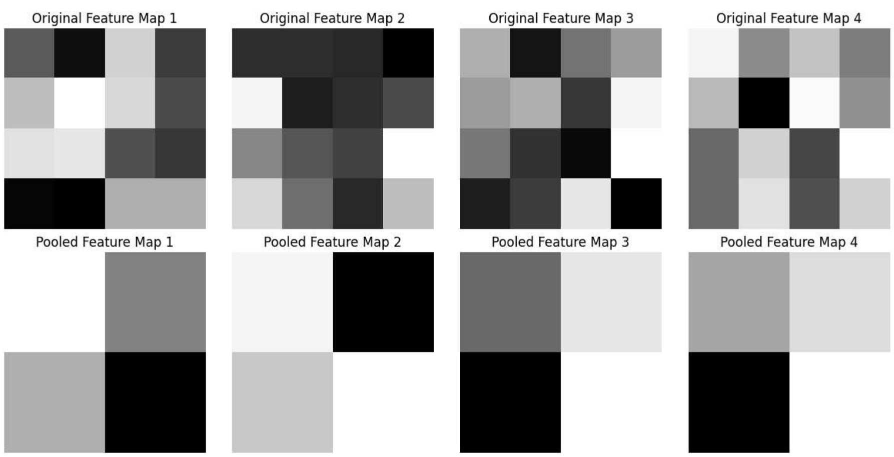
```
Epoch 1/50
18/18 [==============================] - 1s 24ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 2/50
18/18 [==============================] - 0s 8ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 3/50
18/18 [==============================] - 0s 9ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 4/50
18/18 [==============================] - 0s 8ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 5/50
18/18 [==============================] - 0s 6ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 6/50
18/18 [==============================] - 0s 5ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 7/50
18/18 [==============================] - 0s 4ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 8/50
18/18 [==============================] - 0s 5ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 9/50
18/18 [==============================] - 0s 5ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 10/50
18/18 [==============================] - 0s 4ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 11/50
18/18 [==============================] - 0s 5ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 12/50
18/18 [==============================] - 0s 4ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 13/50
18/18 [==============================] - 0s 5ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 14/50
18/18 [==============================] - 0s 5ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 15/50
18/18 [==============================] - 0s 5ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 16/50
18/18 [==============================] - 0s 4ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 17/50
18/18 [==============================] - 0s 4ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 18/50
18/18 [==============================] - 0s 5ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 19/50
18/18 [==============================] - 0s 5ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 20/50
18/18 [==============================] - 0s 5ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 21/50
18/18 [==============================] - 0s 5ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 22/50
18/18 [==============================] - 0s 4ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 23/50
18/18 [==============================] - 0s 4ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 24/50
18/18 [==============================] - 0s 4ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 25/50
18/18 [==============================] - 0s 5ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 26/50
18/18 [==============================] - 0s 5ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 27/50
18/18 [==============================] - 0s 4ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 28/50
18/18 [==============================] - 0s 4ms/step - loss: 26.0175 - val_loss: 24.4711
Epoch 29/50
18/18 [==============================] - 0s 5ms/step - loss: 26.0175 - val_loss: 24.4711
```

In convolutional neural networks, max pooling is a downsampling technique that is frequently used to minimize the spatial dimensions of feature maps while preserving the most crucial information. The input feature map is divided into rectangular, non-overlapping parts, and only the maximum value from each zone is kept in the process.

```
# Define the model to output feature maps
layer_outputs = [model.layers[0].output]  # Output of the first convolutional layer

feature_map_model = Model(inputs=model.input, outputs=layer_outputs)

# Use this model to predict on a sample input to get the feature maps
feature_maps = feature_map_model.predict(X_train[:1])  # Using the first image in the train set

# Feature maps from the first convolutional layer
feature_maps_from_first_layer = feature_maps[0]
```

```
1/1 [==============================] - 0s 146ms/step
```

```
# Import the necessary modules
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# Define the max_pooling function using TensorFlow
def max_pooling(feature_map, pool_size=(2, 2)):
    pooled_feature_map = tf.keras.layers.MaxPooling2D(pool_size=pool_size)(feature_map)
    return pooled_feature_map.numpy()

# Perform max pooling on the random feature map
pooled_feature_map = max_pooling(random_feature_map)

# Visualize the original and pooled feature maps
num_filters_to_visualize = 4
fig, axes = plt.subplots(2, num_filters_to_visualize, figsize=(12, 6))
for i in range(num_filters_to_visualize):
    # Original feature map
    axes[0, i].imshow(random_feature_map[0, :, :, i], cmap='gray')
    axes[0, i].set_title(f'Original Feature Map {i+1}')
    axes[0, i].axis('off')
    # Pooled feature map
    axes[1, i].imshow(pooled_feature_map[0, :, :, i], cmap='gray')
    axes[1, i].set_title(f'Pooled Feature Map {i+1}')
    axes[1, i].axis('off')
plt.tight_layout()
plt.show()
```



Effect of Maxpooling with feature maps

```
import matplotlib.pyplot as plt

def plot_feature_maps(feature_maps, title):
    num_maps = feature_maps.shape[-1]  # Get the number of feature maps
    fig, axes = plt.subplots(1, num_maps, figsize=(num_maps * 2.5, 3))  # Dynamic sizing of the plot
    fig.suptitle(title)

    if num_maps == 1:  # If there is only one feature map, axes is not an array
        axes = [axes]

    for i, ax in enumerate(axes):
        # Displaying the i-th feature map
        ax.imshow(feature_maps[0, :, :, i], cmap='gray', aspect='auto')
        ax.axis('off')
    plt.show()

# Now using the function to plot the feature maps
# Plot feature maps before and after pooling
plot_feature_maps(feature_maps_from_first_layer, "Feature Maps from First Convolutional Layer")
plot_feature_maps(feature_maps_from_second_layer, "Feature Maps from Second Convolutional Layer")
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-39-966d10b58f1e> in <cell line: 19>()
     17 # Now using the function to plot the feature maps
     18 # Plot feature maps before and after pooling
---> 19 plot_feature_maps(feature_maps_from_first_layer, "Feature Maps from First
Convolutional Layer")
     20 plot_feature_maps(feature_maps_from_second_layer, "Feature Maps from Second
Convolutional Layer")

<ipython-input-39-966d10b58f1e> in plot_feature_maps(feature_maps, title)
     11     for i, ax in enumerate(axes):
     12         # Displaying the i-th feature map
---> 13         ax.imshow(feature_maps[0, :, :, i], cmap='gray', aspect='auto')
     14         ax.axis('off')
     15     plt.show()

IndexError: too many indices for array: array is 1-dimensional, but 4 were indexed
```

Next steps: Explain error

```
def plot_feature_maps(feature_maps, title):
    # Check if the input array is 4-dimensional
    if len(feature_maps.shape) != 4:
        raise ValueError("Input array must be 4-dimensional.")

    num_maps = feature_maps.shape[-1]
    fig, axes = plt.subplots(1, num_maps, figsize=(num_maps * 2.5, 3))
    fig.suptitle(title)

    if num_maps == 1:
        axes = [axes]

    for i, ax in enumerate(axes):
        ax.imshow(feature_maps[0, :, :, i], cmap='gray', aspect='auto')
        ax.axis('off')
    plt.show()
```

```
print(feature_maps_from_first_layer.shape)
```

```
(64,)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(8, 8, 1), padding='same'),
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),  # Reduce dimension to 4x4
    Conv2D(128, (3, 3), activation='relu', padding='same'),  # Using padding to maintain dimension
    Flatten(),  # Flattening the outputs from the convolutional layers
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')  # Output layer with softmax activation for 10 classes
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 8, 8, 32)          320

 conv2d_1 (Conv2D)           (None, 8, 8, 64)          18496

 max_pooling2d_1 (MaxPoolin  (None, 4, 4, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 128)         73856

 flatten_1 (Flatten)         (None, 2048)              0

 dense (Dense)               (None, 128)               262272

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 356234 (1.36 MB)
Trainable params: 356234 (1.36 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

K-Fold

```
# Train the model
print(f'Training for fold {fold_no}...')
history = model.fit(X_train_fold, y_train[train_idx],
                        batch_size=32, epochs=10, validation_data=(X_test_fold, y_train[test_idx]))

# Saving scores
scores = model.evaluate(X_test_fold, y_train[test_idx], verbose=0)
print(f'Score for fold {fold_no}: {model.metrics_names[1]} of {scores[1]*100}%')
accuracies.append(scores[1])
losses.append(scores[0])
```

```
Training for fold 1...
Epoch 1/10
36/36 [==============================] - 3s 33ms/step - loss: 1.8040 - accuracy: 0.3934 - val_loss: 0.7367 - val_accuracy: 0.8507
Epoch 2/10
36/36 [==============================] - 1s 37ms/step - loss: 0.6938 - accuracy: 0.7842 - val_loss: 0.2435 - val_accuracy: 0.9549
Epoch 3/10
36/36 [==============================] - 1s 34ms/step - loss: 0.3415 - accuracy: 0.8938 - val_loss: 0.1251 - val_accuracy: 0.9688
Epoch 4/10
36/36 [==============================] - 1s 21ms/step - loss: 0.2724 - accuracy: 0.9138 - val_loss: 0.0897 - val_accuracy: 0.9861
Epoch 5/10
36/36 [==============================] - 1s 22ms/step - loss: 0.1657 - accuracy: 0.9504 - val_loss: 0.0626 - val_accuracy: 0.9861
Epoch 6/10
36/36 [==============================] - 1s 24ms/step - loss: 0.1256 - accuracy: 0.9634 - val_loss: 0.0625 - val_accuracy: 0.9757
Epoch 7/10
36/36 [==============================] - 1s 23ms/step - loss: 0.0775 - accuracy: 0.9791 - val_loss: 0.0334 - val_accuracy: 0.9896
Epoch 8/10
36/36 [==============================] - 1s 23ms/step - loss: 0.0771 - accuracy: 0.9713 - val_loss: 0.0476 - val_accuracy: 0.9792
Epoch 9/10
36/36 [==============================] - 1s 23ms/step - loss: 0.0799 - accuracy: 0.9782 - val_loss: 0.0400 - val_accuracy: 0.9931
Epoch 10/10
36/36 [==============================] - 1s 21ms/step - loss: 0.0764 - accuracy: 0.9756 - val_loss: 0.0270 - val_accuracy: 0.9931
Score for fold 1: accuracy of 99.30555820465088%
```

Evaluation

```
X_test = X_test.reshape(-1, 8, 8, 1)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test)

# Print the results
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
12/12 [==============================] - 0s 12ms/step - loss: 0.0380 - accuracy: 0.9861
Test Loss: 0.037978000938892365
Test Accuracy: 0.9861111044883728
```

Model is evaluated on the test set. Test loss and accuracy are reported.

```
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
conf_matrix = confusion_matrix(y_test, y_pred_classes)
print("Confusion Matrix:")
print(conf_matrix)
```

```
12/12 [==============================] - 49s 6ms/step
Confusion Matrix:
[[33  0  0  0  0  0  0  0  0  0]
 [ 0 28  0  0  0  0  0  0  0  0]
 [ 0  0 33  0  0  0  0  0  0  0]
 [ 0  0  0 33  0  1  0  0  0  0]
 [ 0  0  0  0 46  0  0  0  0  0]
 [ 0  0  0  0  0 47  0  0  0  0]
 [ 0  0  0  0  0  1 34  0  0  0]
 [ 0  0  0  0  0  0  0 34  0  0]
 [ 0  2  0  0  0  0  0  0 28  0]
 [ 0  0  0  0  0  0  0  0  1 39]]
```

```
import matplotlib.pyplot as plt

# Plot training history
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```