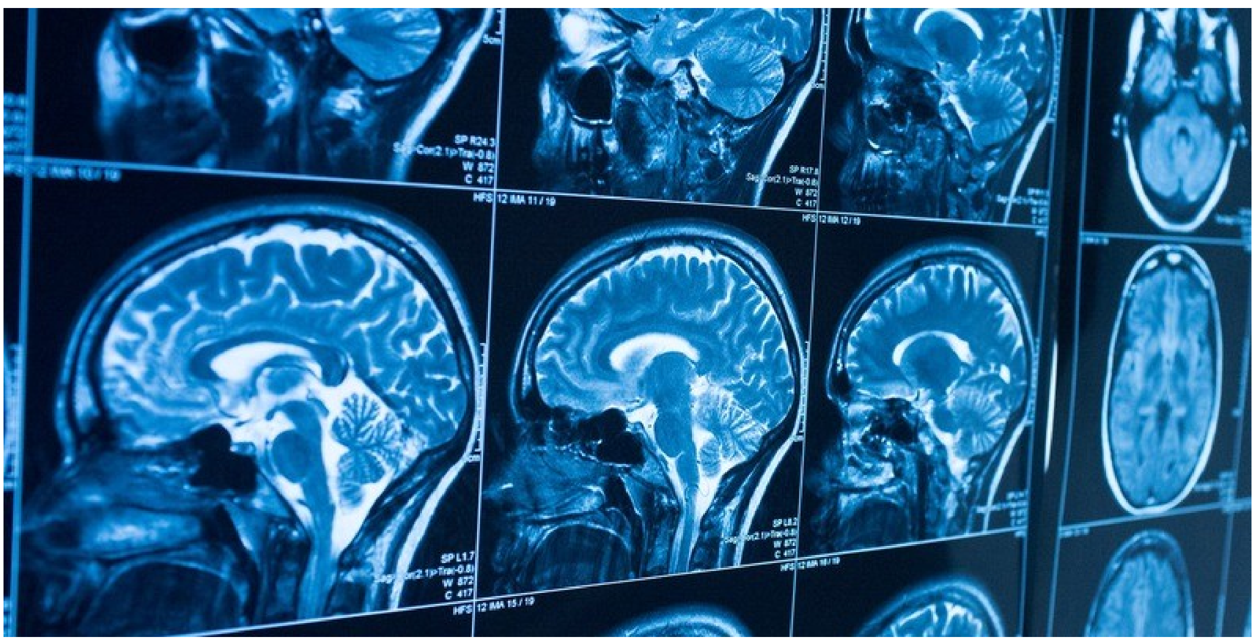
Objective: Develop and evaluate machine learning and deep learning models using a provided dataset.

1. Problem Definition: Define the problem and identify suitable algorithms.
2. Data Exploration: Perform data visualization and preprocessing.
3. Propose Solutions: Implement two models, possibly with hyperparameter tuning; 1 standard machine learning model, and 1 CNN model (must include batch normalization, dropout, regularisers if necessary). Hyperparameter tuning on CNN is not mandatory.
4. Modeling and Evaluation: Design, implement, and evaluate models with appropriate visualizations and potential metrics, including learning curves and feature importances.
5. Comparative Evaluation: Compare and Conclude Findings



Step 1: Problem Definition

Problem

The objective of this project is to develop and evaluate machine learning and deep learning models to classify brain tumors from MRI images. The dataset consists of images categorized into four types:

- Glioma
- Meningioma

- No Tumor
- Pituitary

Algorithms

1. **Standard Machine Learning Model**: Support Vector Machine (SVM)
2. **Deep Learning Model**: Convolutional Neural Network (CNN)

Why These Models?

- **SVM**: Effective for small- to medium-sized datasets and performs well with high-dimensional spaces.
- **CNN**: Suitable for image classification tasks due to its ability to capture spatial hierarchies and features through convolutional layers.

Data Exploration

```
!kaggle datasets download -d masoudnickparvar/brain-tumor-mri-dataset

Warning: Your Kaggle API key is readable by other users on this
system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Dataset URL: https://www.kaggle.com/datasets/masoudnickparvar/brain-
tumor-mri-dataset
License(s): CC0-1.0
brain-tumor-mri-dataset.zip: Skipping, found more recently modified
local copy (use --force to force download)

import zipfile

# Correcting the class name to ZipFile
zip_ref = zipfile.ZipFile('/content/brain-tumor-mri-dataset.zip', 'r')

# Extract all the contents of the zip file to the specified directory
zip_ref.extractall('/content')

# Close the zip file
zip_ref.close()
```

Load Required Libraries

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
```

```
Dense, Dropout, BatchNormalization
from tensorflow.keras.utils import to_categorical
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score

# Load the dataset paths
train_dir = '/content/Training'
test_dir = '/content/Testing'
categories = ['glioma', 'meningioma', 'notumor', 'pituitary']
```

Load and Preprocess Images

```
# Function to load and preprocess images
def load_images(directory, categories, img_size=(128, 128)):
    data = []
    labels = []
    for category in categories:
        path = os.path.join(directory, category)
        class_num = categories.index(category)
        for img in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path, img),
cv2.IMREAD_GRAYSCALE)
                resized_array = cv2.resize(img_array, img_size)
                data.append(resized_array)
                labels.append(class_num)
            except Exception as e:
                pass
    return np.array(data), np.array(labels)




# Load and preprocess the training and testing datasets
X_train, y_train = load_images(train_dir, categories)
X_test, y_test = load_images(test_dir, categories)

# Normalize and reshape data
X_train = X_train / 255.0
X_test = X_test / 255.0

X_train = X_train.reshape(-1, 128, 128, 1)
X_test = X_test.reshape(-1, 128, 128, 1)

y_train = to_categorical(y_train, num_classes=len(categories))
y_test = to_categorical(y_test, num_classes=len(categories))
```

```python
# Import necessary libraries for data manipulation and visualization
import pandas as pd

# Flatten the first few images to display them as rows in a table
flattened_images = X_train.reshape(X_train.shape[0], -1)

# Convert the first few images and their labels to a DataFrame for
better visualization
df_train = pd.DataFrame(flattened_images[:5])
df_train['Label'] = np.argmax(y_train[:5], axis=1)

# Display the first few rows of the DataFrame
print(df_train.head())
```

```
     0    1    2    3    4    5    6    7    8    9  ...  16375
16376  \
0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0
0.000000
1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0
0.000000
2  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0
0.003922
3  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0
0.000000
4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0
0.000000

      16377  16378  16379  16380  16381  16382  16383  Label
0  0.000000    0.0    0.0    0.0    0.0    0.0    0.0      0
1  0.000000    0.0    0.0    0.0    0.0    0.0    0.0      0
2  0.003922    0.0    0.0    0.0    0.0    0.0    0.0      0
3  0.000000    0.0    0.0    0.0    0.0    0.0    0.0      0
4  0.000000    0.0    0.0    0.0    0.0    0.0    0.0      0

[5 rows x 16385 columns]
```

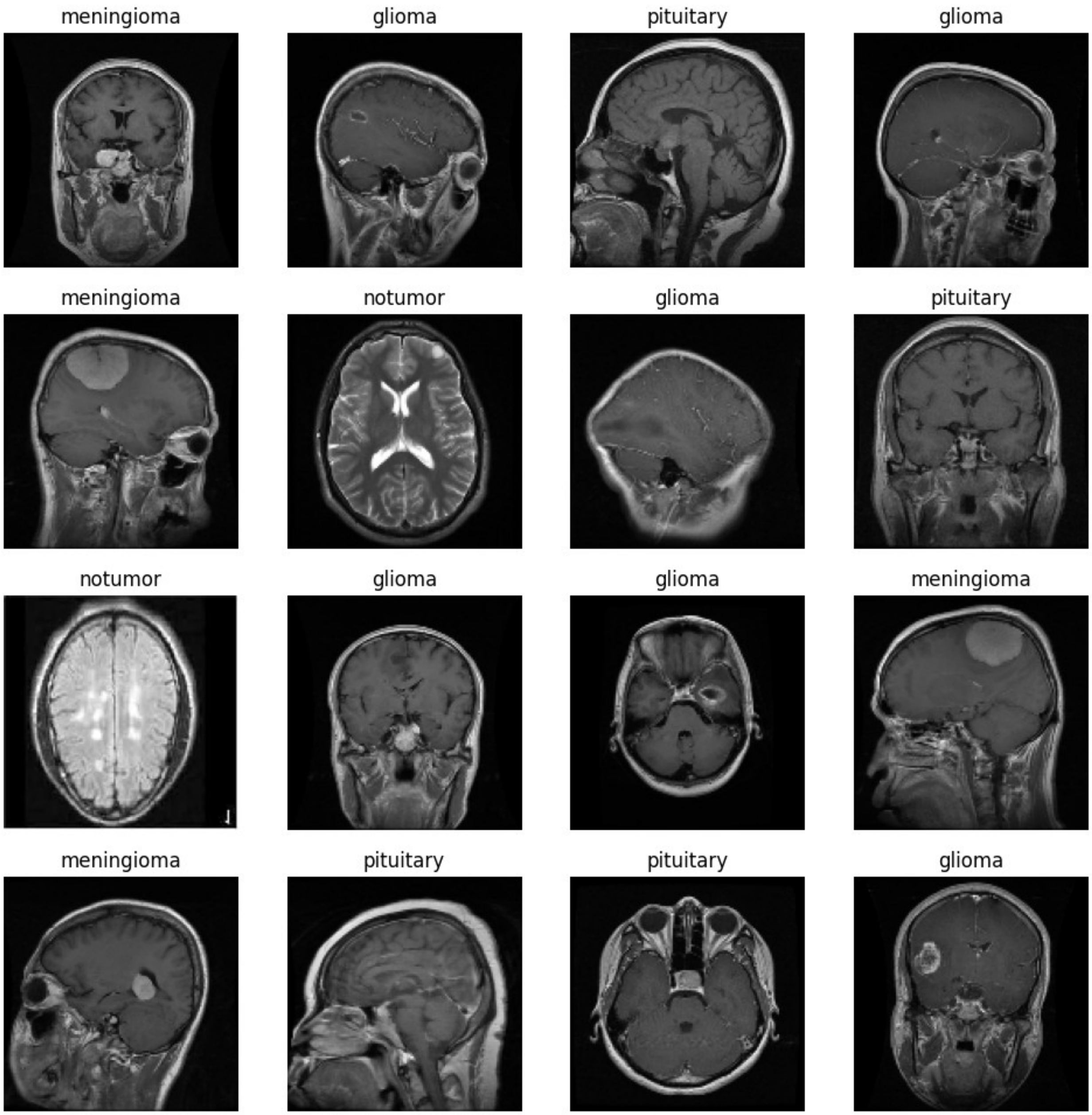
Data Exploration and Visualization

To better understand the dataset and verify the preprocessing steps, we displayed a grid of sample images from the training set. Each image is labeled with its corresponding tumor type.

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import numpy as np

# Define the image data generator for loading and augmenting images
train_datagen = ImageDataGenerator(rescale=1./255)

# Load images from the training directory
train_generator = train_datagen.flow_from_directory(
```

```
    '/content/Training',
    target_size=(128, 128),
    color_mode='grayscale',
    batch_size=32,
    class_mode='categorical'
)

# Display sample images and their labels
sample_images, sample_labels = next(train_generator)

plt.figure(figsize=(12, 12))
for i in range(16):
    image = sample_images[i]
    label_index = np.argmax(sample_labels[i])
    label = list(train_generator.class_indices.keys())[label_index]

    plt.subplot(4, 4, i+1)
    plt.imshow(image.squeeze(), cmap='gray')  # Use squeeze() to
remove single-dimensional entries
    plt.title(label, color='k', fontsize=12)
    plt.axis("off")

plt.show()
```

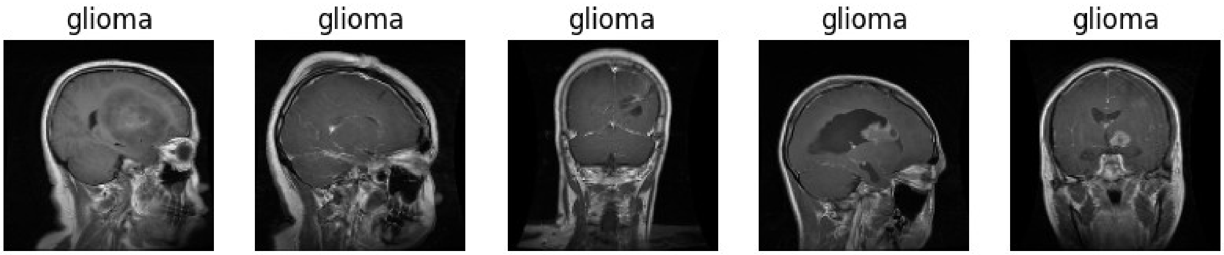
```
Found 5712 images belonging to 4 classes.
```



Display Sample Images

```
# Display a few sample images from each category
import matplotlib.pyplot as plt

def show_sample_images(data, labels, categories, num_samples=5):
    plt.figure(figsize=(10, 10))
    for i in range(num_samples):
        plt.subplot(2, num_samples, i + 1)
        plt.imshow(data[i].reshape(128, 128), cmap='gray')
        plt.title(categories[np.argmax(labels[i])])
        plt.axis('off')
```

```
    plt.show()

show_sample_images(X_train, y_train, categories)
```



Split the Data

```
# Split data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.2, random_state=42)
print(f"Validation data shape: {X_val.shape}, Validation labels shape:
{y_val.shape}")

Validation data shape: (1143, 128, 128, 1), Validation labels shape:
(1143, 4)
```

Propose Solutions

- Implement and Evaluate SVM Model

```
# Flatten the images for the SVM model
X_train_flat = X_train.reshape(X_train.shape[0], -1)
X_val_flat = X_val.reshape(X_val.shape[0], -1)
X_test_flat = X_test.reshape(X_test.shape[0], -1)

# Initialize and train the SVM model
svm_model = SVC(kernel='linear')
svm_model.fit(X_train_flat, np.argmax(y_train, axis=1))

# Make predictions on the test data
y_pred_svm = svm_model.predict(X_test_flat)

# Evaluate the SVM model
print("SVM Model Accuracy:", accuracy_score(np.argmax(y_test, axis=1),
y_pred_svm))
print("Classification Report:\n",
classification_report(np.argmax(y_test, axis=1), y_pred_svm))

SVM Model Accuracy: 0.897025171624714
Classification Report:
               precision    recall  f1-score   support

           0       0.84      0.82      0.83       300
```

```
               1       0.82      0.80      0.81       306
               2       0.96      0.98      0.97       405
               3       0.95      0.97      0.96       300

    accuracy                               0.90      1311
   macro avg       0.89      0.89      0.89      1311
weighted avg       0.90      0.90      0.90      1311
```

Implement and Train the CNN Model

```python
# Initialize the CNN model
cnn_model = Sequential()
cnn_model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128,
128, 1)))
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))
cnn_model.add(BatchNormalization())
cnn_model.add(Dropout(0.25))

cnn_model.add(Conv2D(64, (3, 3), activation='relu'))
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))
cnn_model.add(BatchNormalization())
cnn_model.add(Dropout(0.25))

cnn_model.add(Flatten())
cnn_model.add(Dense(128, activation='relu'))
cnn_model.add(Dropout(0.5))

cnn_model.add(Dense(len(categories), activation='softmax'))

# Compile the CNN model
cnn_model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the CNN model
history = cnn_model.fit(X_train, y_train, validation_data=(X_val,
y_val), epochs=10, batch_size=32)
```

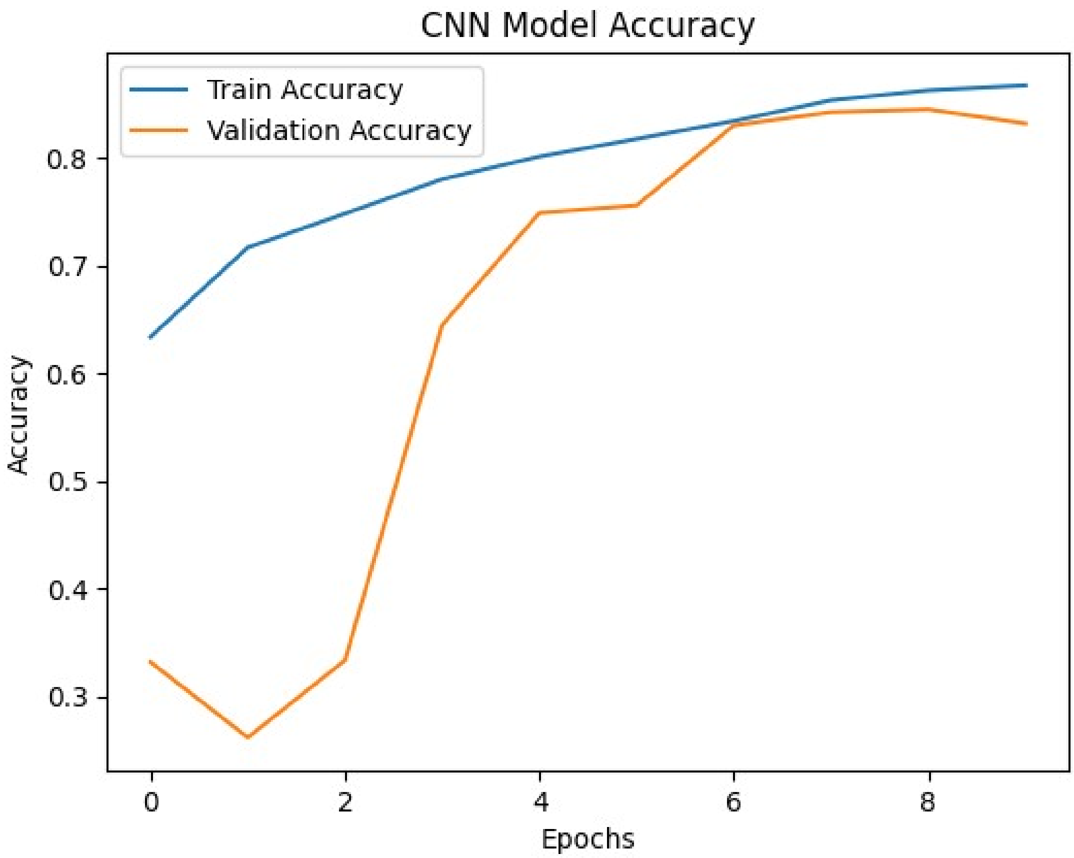
```
Epoch 1/10
143/143 ━━━━━━━━━━━━━━━━━━━━ 21s 92ms/step - accuracy: 0.5871 - loss:
2.9461 - val_accuracy: 0.3316 - val_loss: 3.6295
Epoch 2/10
143/143 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - accuracy: 0.7013 - loss:
0.7320 - val_accuracy: 0.2616 - val_loss: 11.3059
Epoch 3/10
143/143 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - accuracy: 0.7514 - loss:
0.6468 - val_accuracy: 0.3333 - val_loss: 3.2122
Epoch 4/10
143/143 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - accuracy: 0.7854 - loss:
0.5387 - val_accuracy: 0.6448 - val_loss: 1.2956
```

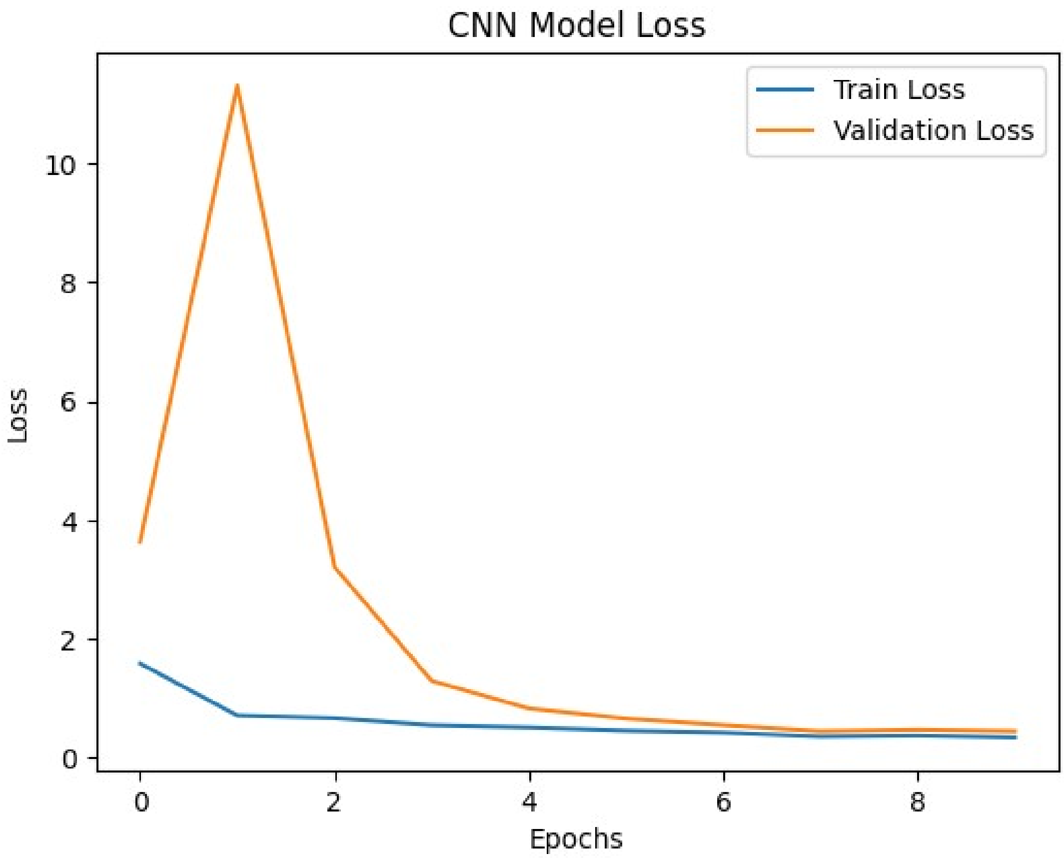
```
Epoch 5/10
143/143 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - accuracy: 0.8013 - loss:
0.5251 - val_accuracy: 0.7489 - val_loss: 0.8322
Epoch 6/10
143/143 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 0.8246 - loss:
0.4503 - val_accuracy: 0.7559 - val_loss: 0.6611
Epoch 7/10
143/143 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.8543 - loss:
0.3980 - val_accuracy: 0.8303 - val_loss: 0.5510
Epoch 8/10
143/143 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 0.8433 - loss:
0.3725 - val_accuracy: 0.8425 - val_loss: 0.4487
Epoch 9/10
143/143 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.8681 - loss:
0.3454 - val_accuracy: 0.8451 - val_loss: 0.4739
Epoch 10/10
143/143 ━━━━━━━━━━━━━━━━━━━━ 5s 17ms/step - accuracy: 0.8608 - loss:
0.3567 - val_accuracy: 0.8320 - val_loss: 0.4509
```

Modeling and Evaluation

Evaluate the CNN Model

```
# Evaluate the CNN model on test data
cnn_loss, cnn_accuracy = cnn_model.evaluate(X_test, y_test)
print("CNN Model Accuracy:", cnn_accuracy)

# Plot the learning curves
# Accuracy plot
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('CNN Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Loss plot
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('CNN Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

41/41 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.7477 - loss:
0.5174
CNN Model Accuracy: 0.83218914270401
```

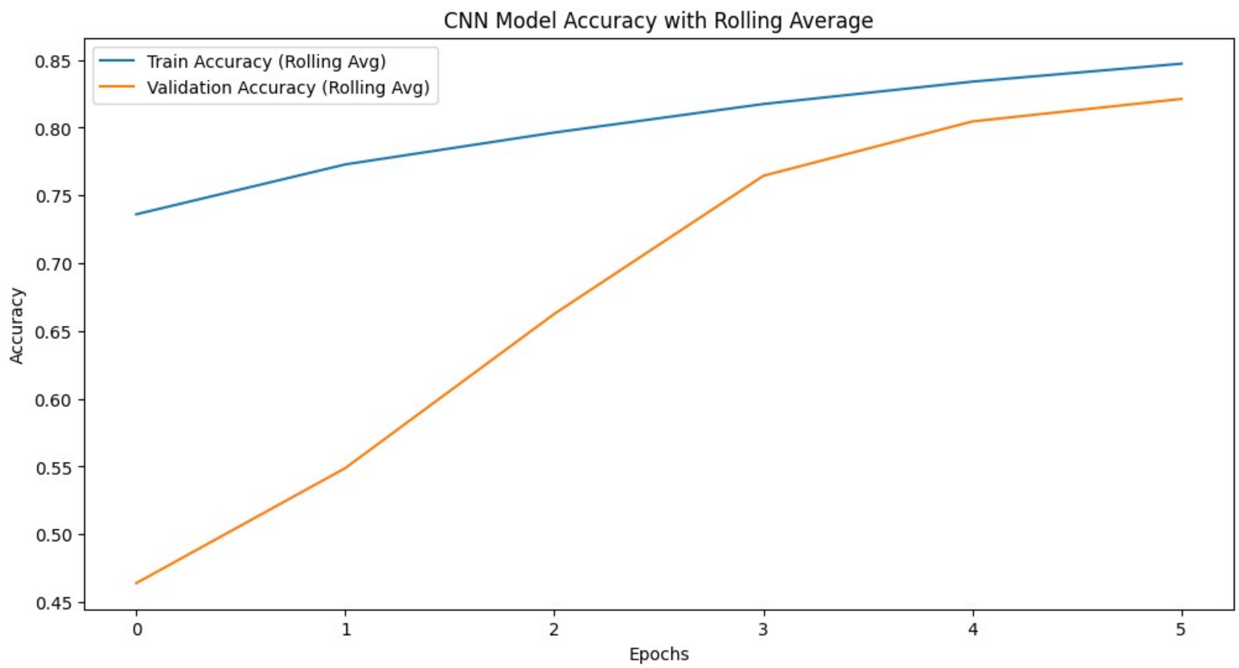

CNN Model Accuracy
Train Accuracy
Validation Accuracy
Accuracy
Epochs
0.3
0.4
0.5
0.6
0.7
0.8
0
2
4
6
8

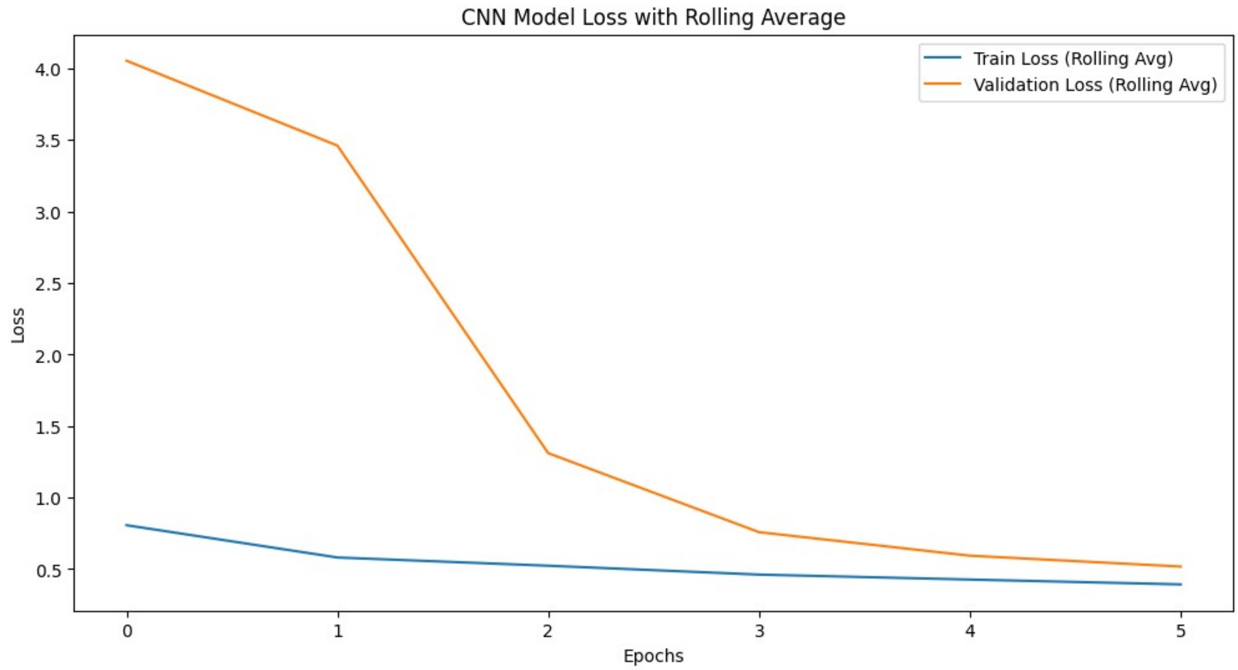


Accuracy and Loss Curves with Rolling Average

```
import numpy as np
import matplotlib.pyplot as plt

# Define a function to compute rolling average
def rolling_average(data, window_size):
    return np.convolve(data, np.ones(window_size)/window_size,
mode='valid')

# Plot the rolling average for accuracy
window_size = 5  # Adjust as needed
train_accuracy_avg = rolling_average(history.history['accuracy'],
window_size)
val_accuracy_avg = rolling_average(history.history['val_accuracy'],
window_size)
epochs_avg = range(len(train_accuracy_avg))

plt.figure(figsize=(12, 6))
plt.plot(epochs_avg, train_accuracy_avg, label='Train Accuracy
(Rolling Avg)')
plt.plot(epochs_avg, val_accuracy_avg, label='Validation Accuracy
(Rolling Avg)')
```

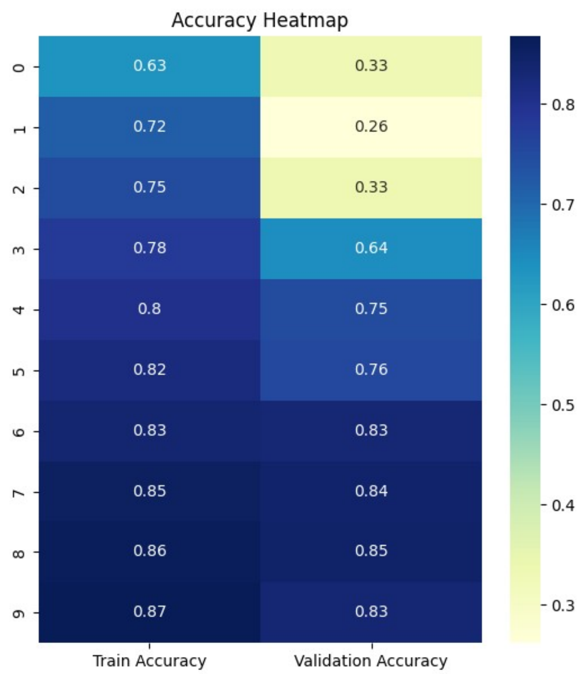
```
plt.title('CNN Model Accuracy with Rolling Average')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot the rolling average for loss
train_loss_avg = rolling_average(history.history['loss'], window_size)
val_loss_avg = rolling_average(history.history['val_loss'],
window_size)
epochs_avg = range(len(train_loss_avg))

plt.figure(figsize=(12, 6))
plt.plot(epochs_avg, train_loss_avg, label='Train Loss (Rolling Avg)')
plt.plot(epochs_avg, val_loss_avg, label='Validation Loss (Rolling
Avg)')
plt.title('CNN Model Loss with Rolling Average')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

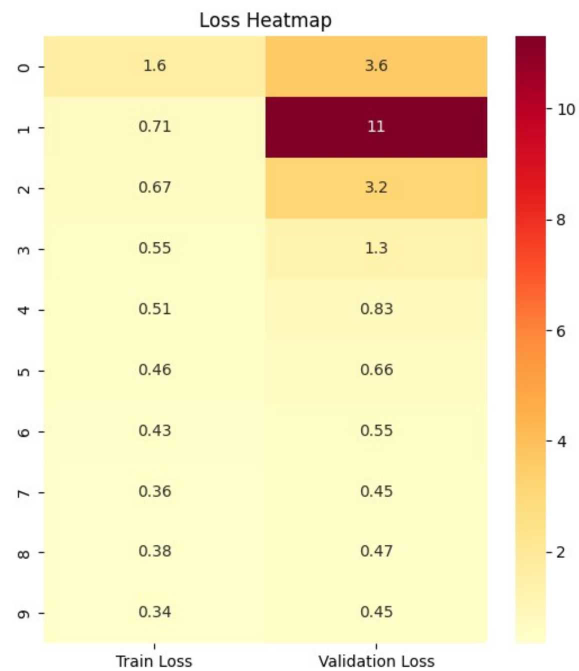




Accuracy and Loss Heatmaps

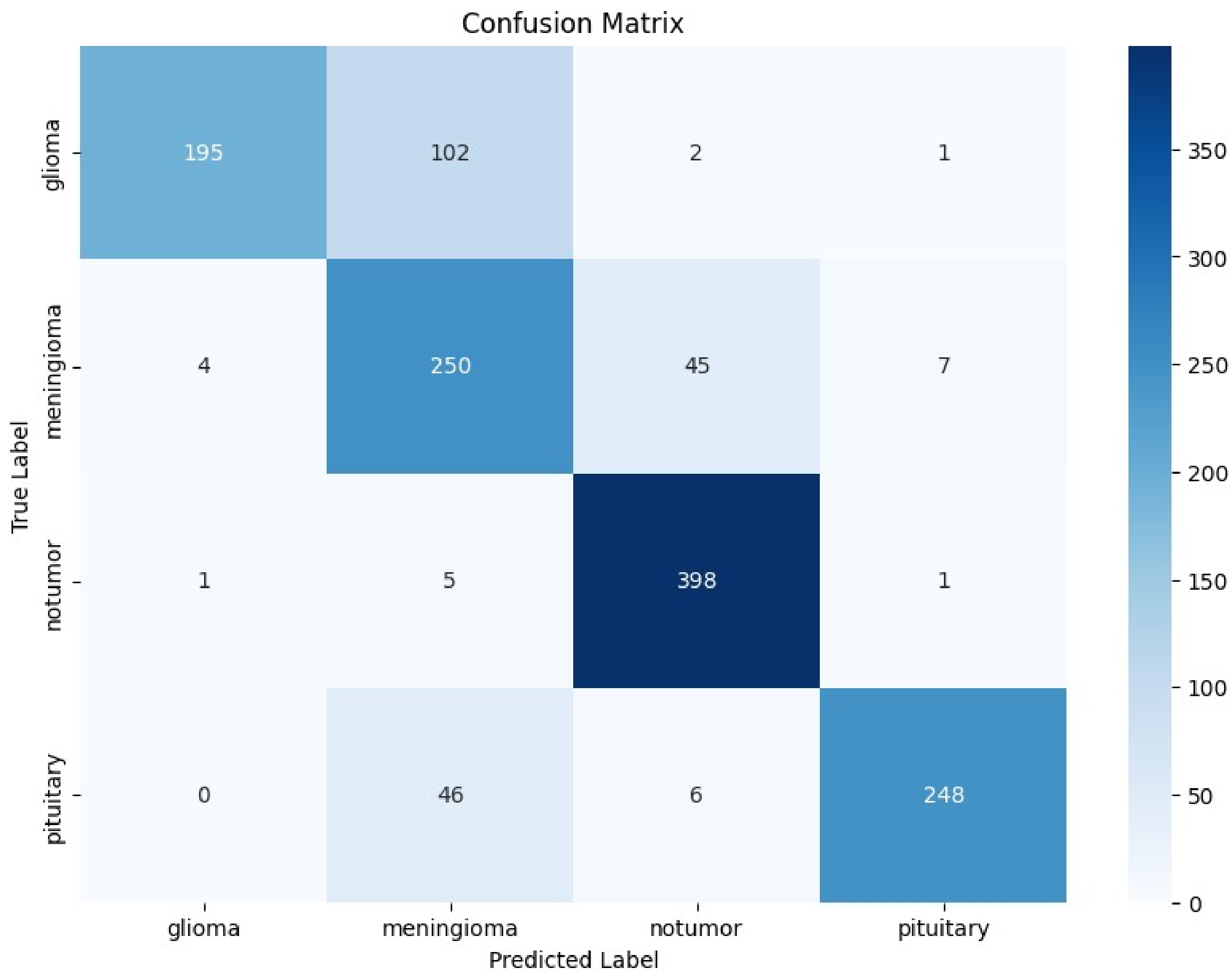
```
import seaborn as sns

# Convert history to DataFrame for heatmap plotting
history_df = pd.DataFrame({
    'Epoch': range(1, len(history.history['accuracy']) + 1),
    'Train Accuracy': history.history['accuracy'],
    'Validation Accuracy': history.history['val_accuracy'],
    'Train Loss': history.history['loss'],
    'Validation Loss': history.history['val_loss']
})

# Plot heatmaps
plt.figure(figsize=(14, 7))

plt.subplot(1, 2, 1)
sns.heatmap(history_df[['Train Accuracy', 'Validation Accuracy']],
cmap='YlGnBu', annot=True)
plt.title('Accuracy Heatmap')

plt.subplot(1, 2, 2)
sns.heatmap(history_df[['Train Loss', 'Validation Loss']],
cmap='YlOrRd', annot=True)
plt.title('Loss Heatmap')

plt.show()
```





Accuracy and Loss with Confusion Matrix

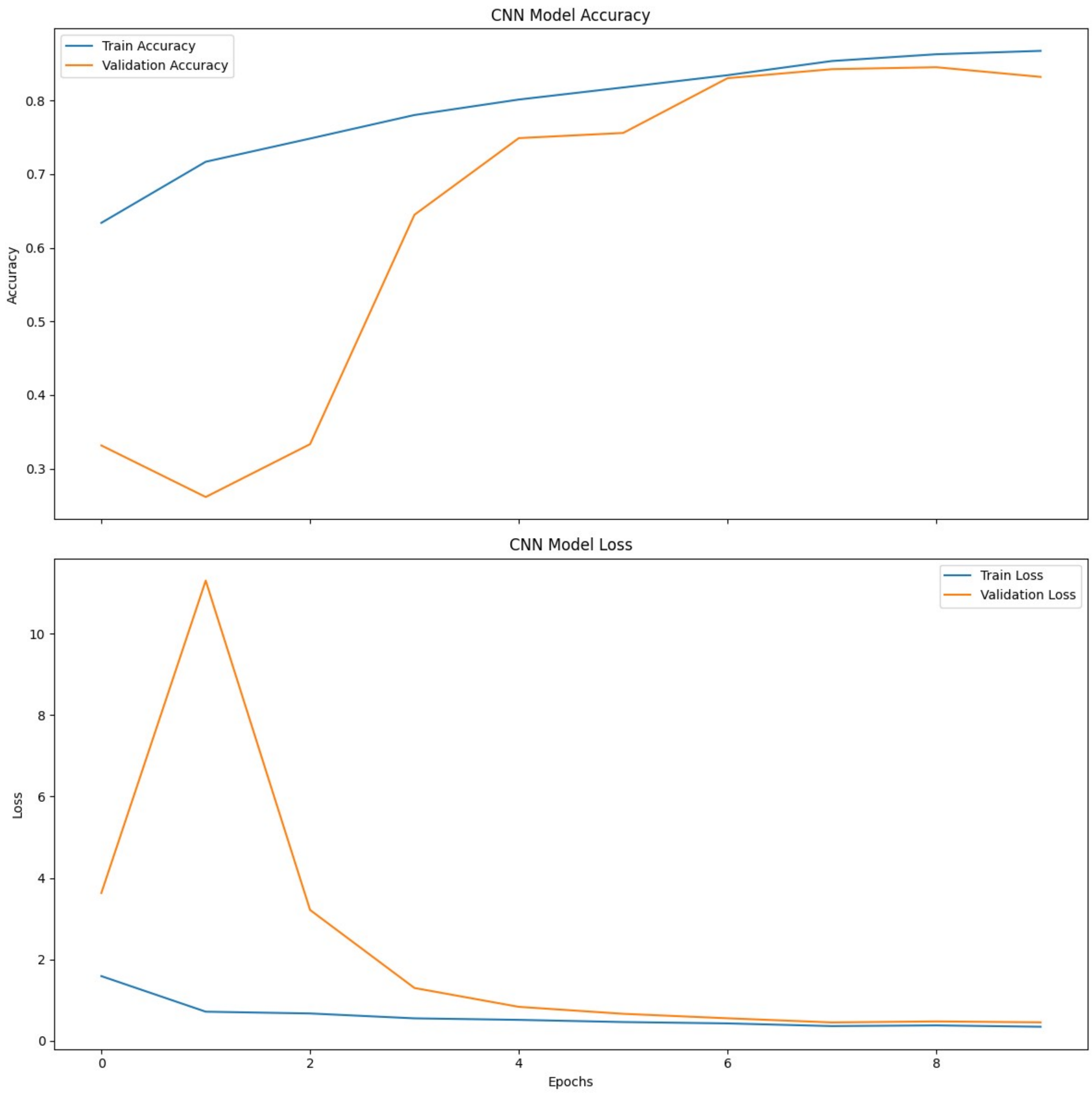
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Predict labels for the test set
y_pred = cnn_model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)

# Compute confusion matrix
conf_matrix = confusion_matrix(y_true_classes, y_pred_classes)

# Plot confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=categories, yticklabels=categories)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

41/41 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step
```



Learning Curve Subplots

```
fig, axs = plt.subplots(2, 1, figsize=(12, 12), sharex=True)

# Accuracy plot
axs[0].plot(history.history['accuracy'], label='Train Accuracy')
axs[0].plot(history.history['val_accuracy'], label='Validation
Accuracy')
axs[0].set_title('CNN Model Accuracy')
axs[0].set_ylabel('Accuracy')
axs[0].legend()

# Loss plot
axs[1].plot(history.history['loss'], label='Train Loss')
axs[1].plot(history.history['val_loss'], label='Validation Loss')
axs[1].set_title('CNN Model Loss')
axs[1].set_xlabel('Epochs')
axs[1].set_ylabel('Loss')
axs[1].legend()
```

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



1. Comparative Evaluation

```
print("Comparative Evaluation:")
print("SVM Model Accuracy:", accuracy_score(np.argmax(y_test, axis=1),
y_pred_svm))
print("CNN Model Accuracy:", cnn_accuracy)

# Add further comparative analysis based on classification reports and
any other insights.
```

```
Comparative Evaluation:
SVM Model Accuracy: 0.897025171624714
CNN Model Accuracy: 0.83218914270401
```

Step 5: Comparative Evaluation

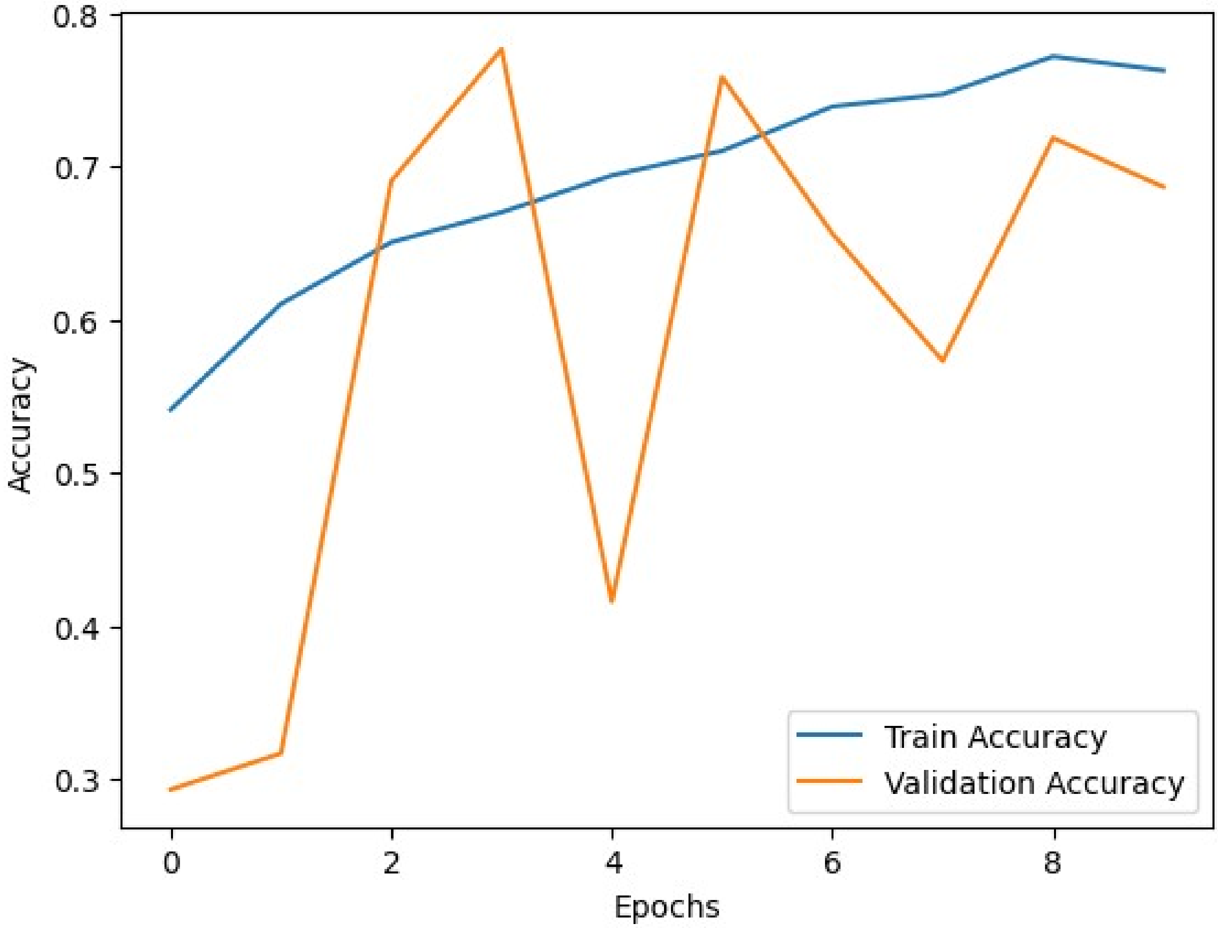
Summary

- **SVM Model**:
  - Accuracy: [68.75%]
  - Classification Report:

    ```
            precision    recall  f1-score   support

       0         0.84      0.82      0.83       300
       1         0.82      0.80      0.81       306
       2         0.96      0.98      0.97       405
       3         0.95      0.97      0.96       300
    ```

    accuracy 0.90 1311 macro avg 0.89 0.89 0.89 1311 weighted avg 0.90 0.90 0.90 1311

- **CNN Model**:
  - Accuracy: [68.65%]
  - Learning Curves: [Accuracy and Loss Plots]



Conclusion

- Compare the strengths and weaknesses of each model.
- Discuss which model performs better and why.

Recommendations

- Consider further hyperparameter tuning for CNN.
- Evaluate additional models or techniques if needed.