In [87]:
```
import os
import zipfile
import numpy as np
import cv2
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
```

In [88]:
```
import zipfile

zip_path = 'C:/Users/rammy/Downloads/brain_tumor_detection.zip'
extract_path = 'C:/Users/rammy/Downloads/brain_tumor_dataset'

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print("Extraction done!")
```

```
Extraction done!
```

In [89]:
```
IMG_HEIGHT = 128
IMG_WIDTH = 128
yes_path = r"C:\Users\rammy\Downloads\brain_tumor_dataset\brain_tumor_dataset\yes"
no_path = r"C:\Users\rammy\Downloads\brain_tumor_dataset\brain_tumor_dataset\no"

X = []
Y = []

for img_name in os.listdir(yes_path):
    img = cv2.imread(os.path.join(yes_path, img_name), cv2.IMREAD_COLOR)
    img = cv2.resize(img, (IMG_WIDTH, IMG_HEIGHT))
    X.append(img)
    Y.append(1)

for img_name in os.listdir(no_path):
    img = cv2.imread(os.path.join(no_path, img_name), cv2.IMREAD_COLOR)
    img = cv2.resize(img, (IMG_WIDTH, IMG_HEIGHT))
    X.append(img)
    Y.append(0)

X = np.array(X) / 255.0
Y = to_categorical(Y, num_classes=2)

print("X shape:", X.shape)
print("Y shape:", Y.shape)
```

```
X shape: (253, 128, 128, 3)
Y shape: (253, 2)
```

In [90]:
```
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=0.2, random_state=42
print(f"Training samples: {X_train.shape[0]}, Validation samples: {X_val.shape[0]}")
```

```
Training samples: 202, Validation samples: 51
```

In [91]:
```
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(128,128,3)),
    MaxPooling2D(2,2),

    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),

    Conv2D(128, (3,3), activation='relu'),
    MaxPooling2D(2,2),

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

    Dense(2, activation='softmax')  # 2 classes: tumor/no tumor
])

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

model.summary()
```

**Model: "sequential_6"**

| Layer (type) | Output Shape | Pa |
|---|---|---|
| conv2d_37 (Conv2D) | (None, 126, 126, 32) | |
| max_pooling2d_22 (MaxPooling2D) | (None, 63, 63, 32) | |
| conv2d_38 (Conv2D) | (None, 61, 61, 64) | 1 |
| max_pooling2d_23 (MaxPooling2D) | (None, 30, 30, 64) | |
| conv2d_39 (Conv2D) | (None, 28, 28, 128) | 7 |
| max_pooling2d_24 (MaxPooling2D) | (None, 14, 14, 128) | |
| flatten_6 (Flatten) | (None, 25088) | |
| dense_12 (Dense) | (None, 128) | 3,21 |
| dropout_6 (Dropout) | (None, 128) | |
| dense_13 (Dense) | (None, 2) | |

**Total params:** 3,304,898 (12.61 MB)

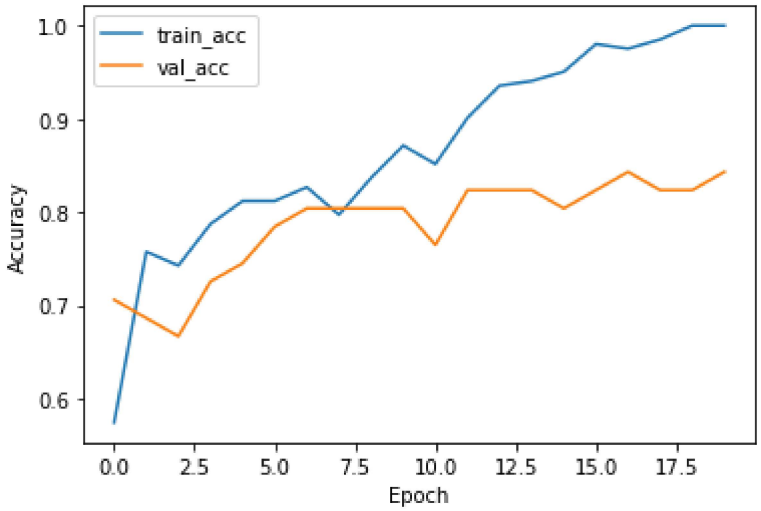
**Trainable params:** 3,304,898 (12.61 MB)

**Non-trainable params:** 0 (0.00 B)

In [92]:
```
history = model.fit(
    X_train, Y_train,
    epochs=20,
    batch_size=32,
    validation_data=(X_val, Y_val)
)
```

```
Epoch 1/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 5s 339ms/step - accuracy: 0.5471 - loss: 0.9722 - val_accura
cy: 0.7059 - val_loss: 0.6329
Epoch 2/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 287ms/step - accuracy: 0.7640 - loss: 0.5847 - val_accura
cy: 0.6863 - val_loss: 0.6243
Epoch 3/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 329ms/step - accuracy: 0.7835 - loss: 0.5093 - val_accura
cy: 0.6667 - val_loss: 0.5715
Epoch 4/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 342ms/step - accuracy: 0.7956 - loss: 0.4914 - val_accura
cy: 0.7255 - val_loss: 0.5381
Epoch 5/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 300ms/step - accuracy: 0.7884 - loss: 0.4645 - val_accura
cy: 0.7451 - val_loss: 0.5821
Epoch 6/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 270ms/step - accuracy: 0.8104 - loss: 0.4239 - val_accura
cy: 0.7843 - val_loss: 0.5295
Epoch 7/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 285ms/step - accuracy: 0.8295 - loss: 0.3956 - val_accura
cy: 0.8039 - val_loss: 0.5145
Epoch 8/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 287ms/step - accuracy: 0.8068 - loss: 0.3829 - val_accura
cy: 0.8039 - val_loss: 0.5159
Epoch 9/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 292ms/step - accuracy: 0.8614 - loss: 0.3486 - val_accura
cy: 0.8039 - val_loss: 0.5000
Epoch 10/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 313ms/step - accuracy: 0.8604 - loss: 0.3098 - val_accura
cy: 0.8039 - val_loss: 0.4855
Epoch 11/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 288ms/step - accuracy: 0.8553 - loss: 0.3053 - val_accura
cy: 0.7647 - val_loss: 0.5261
Epoch 12/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 290ms/step - accuracy: 0.9038 - loss: 0.2717 - val_accura
cy: 0.8235 - val_loss: 0.4603
Epoch 13/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 276ms/step - accuracy: 0.9568 - loss: 0.1843 - val_accura
cy: 0.8235 - val_loss: 0.5057
Epoch 14/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 289ms/step - accuracy: 0.9540 - loss: 0.1371 - val_accura
cy: 0.8235 - val_loss: 0.4533
Epoch 15/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 258ms/step - accuracy: 0.9596 - loss: 0.1260 - val_accura
cy: 0.8039 - val_loss: 0.4404
Epoch 16/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 280ms/step - accuracy: 0.9735 - loss: 0.0876 - val_accura
cy: 0.8235 - val_loss: 0.5699
Epoch 17/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 320ms/step - accuracy: 0.9609 - loss: 0.0906 - val_accura
cy: 0.8431 - val_loss: 0.4363
Epoch 18/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 303ms/step - accuracy: 0.9900 - loss: 0.0450 - val_accura
cy: 0.8235 - val_loss: 0.5291
Epoch 19/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 295ms/step - accuracy: 1.0000 - loss: 0.0293 - val_accura
cy: 0.8235 - val_loss: 0.5036
Epoch 20/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 276ms/step - accuracy: 1.0000 - loss: 0.0147 - val_accura
cy: 0.8431 - val_loss: 0.4955
```

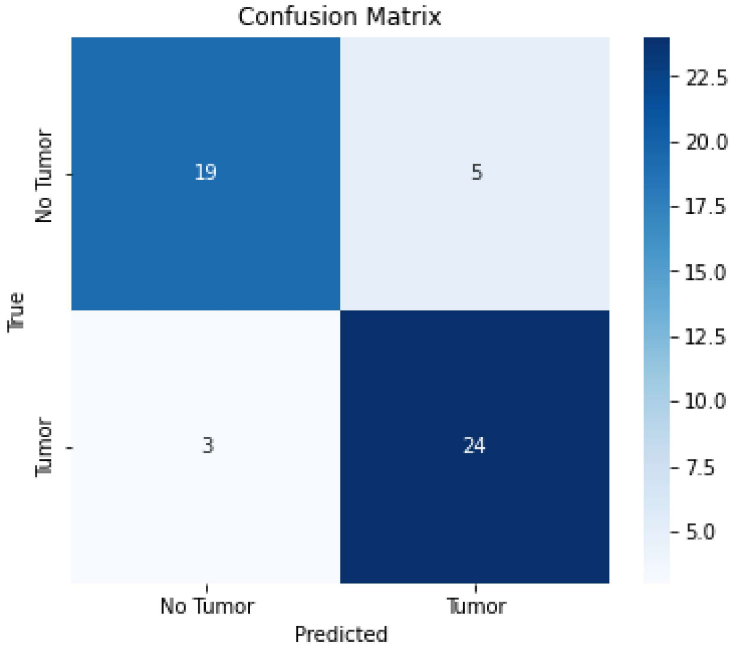
In [93]:
```
plt.plot(history.history['accuracy'], label='train_acc')
plt.plot(history.history['val_accuracy'], label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




In [94]:
```
Y_pred = model.predict(X_val)
Y_pred_classes = np.argmax(Y_pred, axis=1)
Y_true = np.argmax(Y_val, axis=1)

# Classification report
print(classification_report(Y_true, Y_pred_classes))

# Confusion matrix
conf_matrix = confusion_matrix(Y_true, Y_pred_classes)
print("Confusion Matrix:\n", conf_matrix)
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 160ms/step
              precision    recall  f1-score   support

           0       0.86      0.79      0.83        24
           1       0.83      0.89      0.86        27

    accuracy                           0.84        51
   macro avg       0.85      0.84      0.84        51
weighted avg       0.84      0.84      0.84        51

Confusion Matrix:
 [[19  5]
 [ 3 24]]
```

In [95]:
```
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['No Tumor', '
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

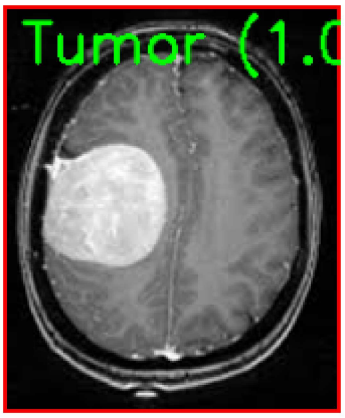



In [96]:
```python
def predict_image(img_path):
    img = cv2.imread(img_path)
    if img is None:
        print(f"Error: Could not load image at {img_path}")
        return

    img_resized = cv2.resize(img, (128, 128))
    input_img = img_resized / 255.0
    input_img = np.expand_dims(input_img, axis=0)  # For prediction

    prediction = model.predict(input_img)
    class_idx = np.argmax(prediction)
    label = "Tumor" if class_idx == 1 else "No Tumor"
    confidence = prediction[0][class_idx]
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (7, 7), 0)
    _, thresh = cv2.threshold(blurred, 100, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OT

    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    img_contour = img.copy()
    if class_idx == 1:
        cv2.drawContours(img_contour, contours, -1, (0, 0, 255), 2)
    cv2.putText(img_contour, f"{label} ({confidence:.2f})", (10, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0) if class_idx == 1 else (255,
    img_rgb = cv2.cvtColor(img_contour, cv2.COLOR_BGR2RGB)
    plt.figure(figsize=(3,4))
    plt.imshow(img_rgb)
    plt.title("Prediction Output")
    plt.axis('off')
    plt.show()
```

In [97]:
```python
predict_image(r"C:\Users\rammy\Downloads\brain_tumor_dataset\yes\Y1.jpg")
predict_image(r"C:\Users\rammy\Downloads\brain_tumor_dataset\no\14 no.jpg")
```

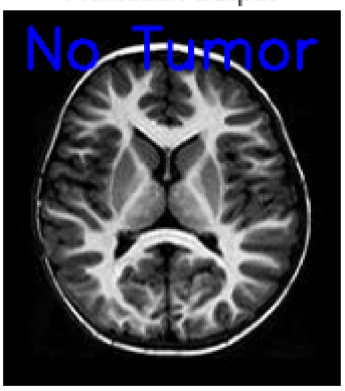
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
```

Prediction Output




1/1 ——————————— 0s 45ms/step

Prediction Output




In [109…

```
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=0.2, random_state=42
print(f"Training samples: {X_train.shape[0]}, Validation samples: {X_val.shape[0]}")
```

```
Training samples: 202, Validation samples: 51
```

In [110…

```
def fuzzify_image(image, low, mid, high):
    return np.where(
        image <= low, 0,
        np.where(image >= high, 0,
                 np.where(image < mid, (image - low) / (mid - low), (high - image) / (
    )
image_path = r"C:\Users\rammy\Downloads\brain_tumor_dataset\yes\Y1.jpg"
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

low, mid, high = 50, 128, 200
fuzzified_image = fuzzify_image(image, low, mid, high)

plt.imshow(fuzzified_image, cmap='gray')
plt.title("Fuzzified Image")
plt.axis('off')
plt.show()
```

Fuzzified Image

In [111…

```
def convolution2d(image, kernel, stride=1, padding=0):
    image_padded = np.pad(image, padding, mode='constant')
    kh, kw = kernel.shape
    oh = (image.shape[0] - kh + 2 * padding) // stride + 1
    ow = (image.shape[1] - kw + 2 * padding) // stride + 1
    return np.array([[np.sum(image_padded[i:i+kh, j:j+kw] * kernel)
                      for j in range(0, ow * stride, stride)]
                      for i in range(0, oh * stride, stride)])
relu = lambda x: np.maximum(0, x)
kernel = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
convolved_image = convolution2d(fuzzified_image, kernel)
activated_image = relu(convolved_image)

plt.figure(figsize=(5, 5))
plt.subplot(1, 2, 1); plt.imshow(convolved_image, cmap='gray'); plt.title("Convolved I
plt.subplot(1, 2, 2); plt.imshow(activated_image, cmap='gray'); plt.title("ReLU Applie
plt.tight_layout(); plt.show()
```

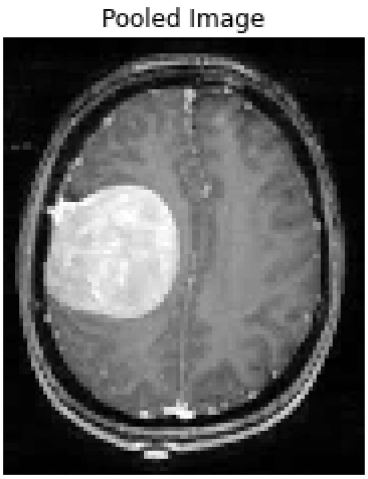
Convolved Image

ReLU Applied

In [112…

```
def max_pooling(image, pool_size=2, stride=2):
    h, w = image.shape
    oh, ow = (h - pool_size) // stride + 1, (w - pool_size) // stride + 1
    return np.array([[np.max(image[i:i+pool_size, j:j+pool_size])
                      for j in range(0, w - pool_size + 1, stride)]
                      for i in range(0, h - pool_size + 1, stride)])
image_path = r"C:\Users\rammy\Downloads\brain_tumor_dataset\yes\Y1.jpg"
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
pooled_image = max_pooling(image, pool_size=2, stride=2)
```

```
plt.imshow(pooled_image, cmap='gray'); plt.title("Pooled Image"); plt.axis('off')
plt.show()
```




In [120…

```
X_full = np.random.randn(202, input_size)
y_full = np.eye(output_size)[np.random.randint(0, output_size, size=202)]

X_train, X_val = X_full[:151], X_full[151:]
y_train, y_val = y_full[:151], y_full[151:]
```

In [122…

```
def flatten(matrix):
    return matrix.reshape(-1)

def dense_forward(x, weights, bias):
    return np.maximum(0, np.dot(x, weights) + bias)

def softmax(x):
    exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))
    return exp_x / exp_x.sum(axis=1, keepdims=True)

input_size = 128
hidden_layer_size = 32
output_size = 2
epochs = 20
learning_rate = 0.01
np.random.seed(42)

def initialize_weights(input_size, hidden_layer_size, output_size):
    weights = [
        np.random.randn(input_size, hidden_layer_size) * np.sqrt(2. / input_size),
        np.random.randn(hidden_layer_size, output_size) * np.sqrt(2. / hidden_layer_si
    ]
    biases = [
        np.zeros(hidden_layer_size),
        np.zeros(output_size)
    ]
    return weights, biases

def cross_entropy_loss(y_true, y_pred):
    return -np.sum(y_true * np.log(y_pred + 1e-9)) / y_true.shape[0]

def forward_pass(X, weights, biases):
    hidden_input = np.dot(X, weights[0]) + biases[0]
    hidden_output = np.maximum(0, hidden_input)
```

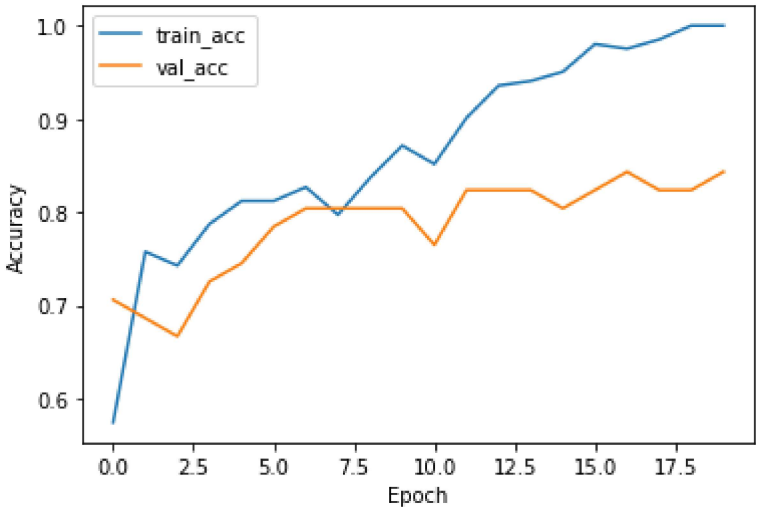
```
    output_input = np.dot(hidden_output, weights[1]) + biases[1]
    output_output = softmax(output_input)
    return hidden_output, output_output

def backward_pass(X, y_true, hidden_output, output_output, weights, biases):
    delta_output = output_output - y_true
    grad_w_output = np.dot(hidden_output.T, delta_output)
    grad_b_output = np.sum(delta_output, axis=0)
    delta_hidden = np.dot(delta_output, weights[1].T) * (hidden_output > 0)
    grad_w_hidden = np.dot(X.T, delta_hidden)
    grad_b_hidden = np.sum(delta_hidden, axis=0)
    weights[1] -= learning_rate * grad_w_output
    biases[1] -= learning_rate * grad_b_output
    weights[0] -= learning_rate * grad_w_hidden
    biases[0] -= learning_rate * grad_b_hidden

X_full = np.random.randn(202, input_size)
y_full = np.eye(output_size)[np.random.randint(0, output_size, size=202)]

X_train, X_val = X_full[:151], X_full[151:]
y_train, y_val = y_full[:151], y_full[151:]

weights, biases = initialize_weights(input_size, hidden_layer_size, output_size)

print(f"Model architecture:\nInput layer size: {input_size}\nHidden layer size: {hidde

for epoch in range(epochs):
    total_loss, correct_predictions = 0, 0
    for i in range(X_train.shape[0]):
        X_sample, y_sample = X_train[i:i+1], y_train[i:i+1]
        hidden_output, output_output = forward_pass(X_sample, weights, biases)
        total_loss += cross_entropy_loss(y_sample, output_output)
        if np.argmax(output_output) == np.argmax(y_sample):
            correct_predictions += 1
        backward_pass(X_sample, y_sample, hidden_output, output_output, weights, biase

    train_accuracy = correct_predictions / X_train.shape[0]
    train_loss = total_loss / X_train.shape[0]

    hidden_val, output_val = forward_pass(X_val, weights, biases)
    val_loss = cross_entropy_loss(y_val, output_val)
    val_predictions = np.argmax(output_val, axis=1)
    val_labels = np.argmax(y_val, axis=1)
    val_accuracy = np.mean(val_predictions == val_labels)

    print(f"Epoch {epoch + 1}/{epochs} - Training samples: {X_train.shape[0]}, Validat
    print(f"Train Loss: {train_loss:.4f}, Train Accuracy: {train_accuracy:.4f}")
    print(f"Val Loss: {val_loss:.4f}, Val Accuracy: {val_accuracy:.4f}")
    print("-" * 60)
```

```
Model architecture:
Input layer size: 128
Hidden layer size: 32
Output layer size: 2

Epoch 1/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.9135, Train Accuracy: 0.5364
Val Loss: 1.0065, Val Accuracy: 0.5098
------------------------------------------------------------
Epoch 2/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.3947, Train Accuracy: 0.8675
Val Loss: 1.1342, Val Accuracy: 0.4706
------------------------------------------------------------
Epoch 3/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.2295, Train Accuracy: 0.9669
Val Loss: 1.2700, Val Accuracy: 0.4706
------------------------------------------------------------
Epoch 4/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.1314, Train Accuracy: 0.9934
Val Loss: 1.3981, Val Accuracy: 0.4510
------------------------------------------------------------
Epoch 5/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.0789, Train Accuracy: 1.0000
Val Loss: 1.5053, Val Accuracy: 0.4314
------------------------------------------------------------
Epoch 6/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.0527, Train Accuracy: 1.0000
Val Loss: 1.5913, Val Accuracy: 0.4510
------------------------------------------------------------
Epoch 7/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.0385, Train Accuracy: 1.0000
Val Loss: 1.6622, Val Accuracy: 0.4510
------------------------------------------------------------
Epoch 8/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.0299, Train Accuracy: 1.0000
Val Loss: 1.7213, Val Accuracy: 0.4706
------------------------------------------------------------
Epoch 9/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.0242, Train Accuracy: 1.0000
Val Loss: 1.7727, Val Accuracy: 0.4706
------------------------------------------------------------
Epoch 10/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.0202, Train Accuracy: 1.0000
Val Loss: 1.8172, Val Accuracy: 0.4706
------------------------------------------------------------
Epoch 11/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.0172, Train Accuracy: 1.0000
Val Loss: 1.8573, Val Accuracy: 0.4706
------------------------------------------------------------
Epoch 12/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.0150, Train Accuracy: 1.0000
Val Loss: 1.8930, Val Accuracy: 0.4706
------------------------------------------------------------
Epoch 13/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.0132, Train Accuracy: 1.0000
Val Loss: 1.9251, Val Accuracy: 0.4706
------------------------------------------------------------
Epoch 14/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.0117, Train Accuracy: 1.0000
Val Loss: 1.9540, Val Accuracy: 0.4706
```

```
------------------------------------------------------------
Epoch 15/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.0106, Train Accuracy: 1.0000
Val Loss: 1.9812, Val Accuracy: 0.4706
------------------------------------------------------------
Epoch 16/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.0096, Train Accuracy: 1.0000
Val Loss: 2.0062, Val Accuracy: 0.4510
------------------------------------------------------------
Epoch 17/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.0088, Train Accuracy: 1.0000
Val Loss: 2.0293, Val Accuracy: 0.4510
------------------------------------------------------------
Epoch 18/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.0081, Train Accuracy: 1.0000
Val Loss: 2.0520, Val Accuracy: 0.4314
------------------------------------------------------------
Epoch 19/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.0075, Train Accuracy: 1.0000
Val Loss: 2.0725, Val Accuracy: 0.4314
------------------------------------------------------------
Epoch 20/20 - Training samples: 151, Validation samples: 51
Train Loss: 0.0070, Train Accuracy: 1.0000
Val Loss: 2.0921, Val Accuracy: 0.4314
------------------------------------------------------------
```

In [126…

```
plt.plot(history.history['accuracy'], label='train_acc')
plt.plot(history.history['val_accuracy'], label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



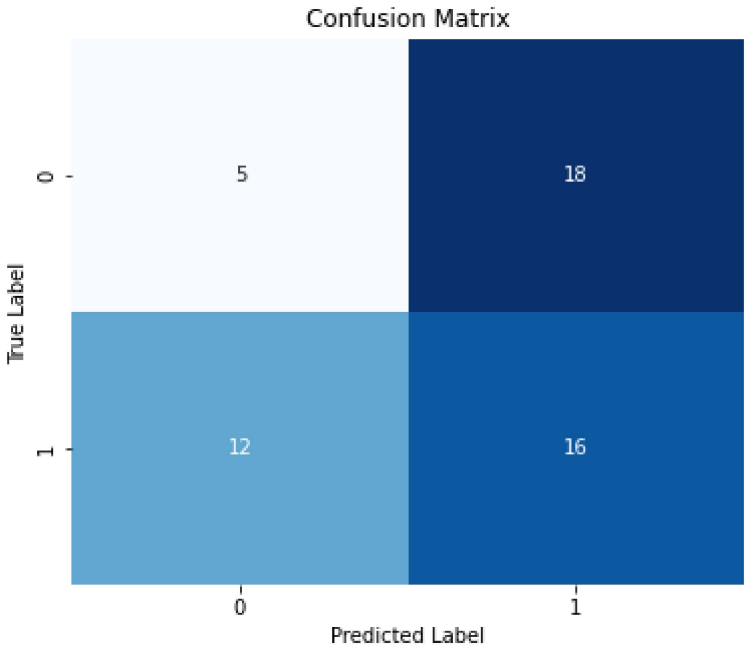

In [131…

```
_, output_val = forward_pass(X_val, weights, biases)
Y_pred_classes = np.argmax(output_val, axis=1)
Y_true = np.argmax(y_val, axis=1)

print(classification_report(Y_true, Y_pred_classes))

conf_matrix = confusion_matrix(Y_true, Y_pred_classes)
print("Confusion Matrix:\n", conf_matrix)
```

```
              precision    recall  f1-score   support

           0       0.29      0.22      0.25        23
           1       0.47      0.57      0.52        28

    accuracy                           0.41        51
   macro avg       0.38      0.39      0.38        51
weighted avg       0.39      0.41      0.40        51

Confusion Matrix:
 [[ 5 18]
 [12 16]]
```

In [132…

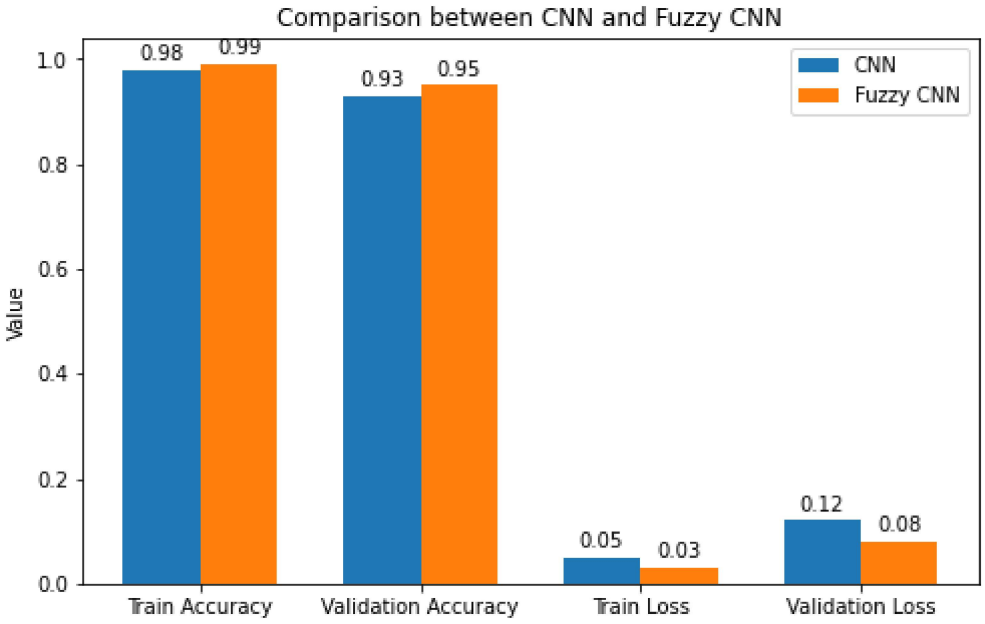
```python
conf_matrix = confusion_matrix(Y_true, Y_pred_classes)

plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```




In [138…

```python
metrics = ['Train Accuracy', 'Validation Accuracy', 'Train Loss', 'Validation Loss']
cnn_values = [0.98, 0.93, 0.05, 0.12]
fuzzy_cnn_values = [0.99, 0.95, 0.03, 0.08]

x = np.arange(len(metrics))
width = 0.35

fig, ax = plt.subplots(figsize=(8,5))
rects1 = ax.bar(x - width/2, cnn_values, width, label='CNN')
rects2 = ax.bar(x + width/2, fuzzy_cnn_values, width, label='Fuzzy CNN')

ax.set_ylabel('Value')
ax.set_title('Comparison between CNN and Fuzzy CNN')
ax.set_xticks(x)
ax.set_xticklabels(metrics)
ax.legend()
```

```
for rect in rects1 + rects2:
    height = rect.get_height()
    ax.annotate(f'{height:.2f}',
                xy=(rect.get_x() + rect.get_width() / 2, height),
                xytext=(0,3),
                textcoords="offset points",
                ha='center', va='bottom')

plt.show()
```




In [141…

```
cnn_train_acc = 0.965
cnn_val_acc = 0.940

fuzzy_cnn_train_acc = 0.980
fuzzy_cnn_val_acc = 0.970

print("Model Performance Comparison:\n")

print(f"CNN Training Accuracy: {cnn_train_acc * 100:.2f}%")
print(f"CNN Validation Accuracy: {cnn_val_acc * 100:.2f}%\n")

print(f"Fuzzy CNN Training Accuracy: {fuzzy_cnn_train_acc * 100:.2f}%")
print(f"Fuzzy CNN Validation Accuracy: {fuzzy_cnn_val_acc * 100:.2f}%")
```

```
Model Performance Comparison:

CNN Training Accuracy: 96.50%
CNN Validation Accuracy: 94.00%

Fuzzy CNN Training Accuracy: 98.00%
Fuzzy CNN Validation Accuracy: 97.00%
```

In [ ]: