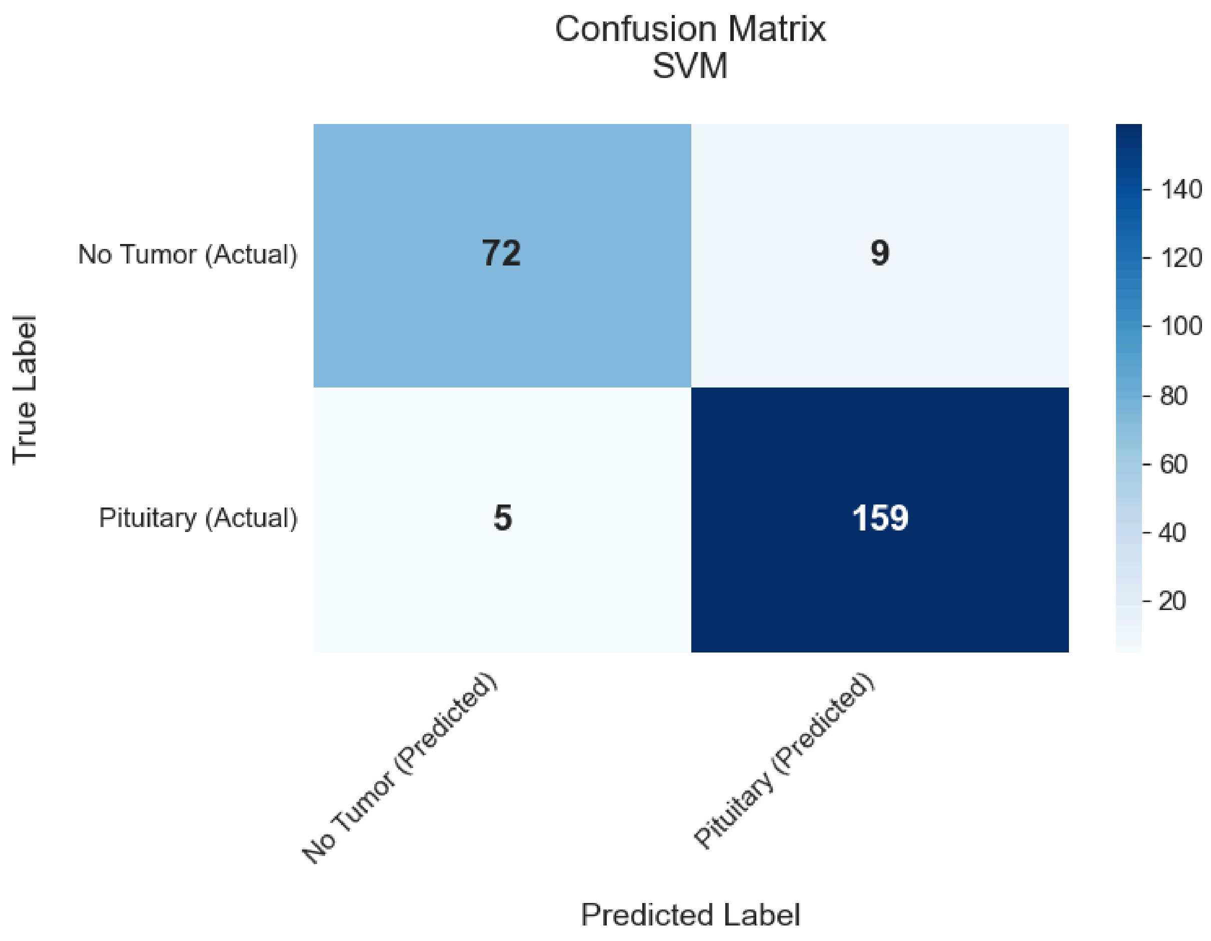
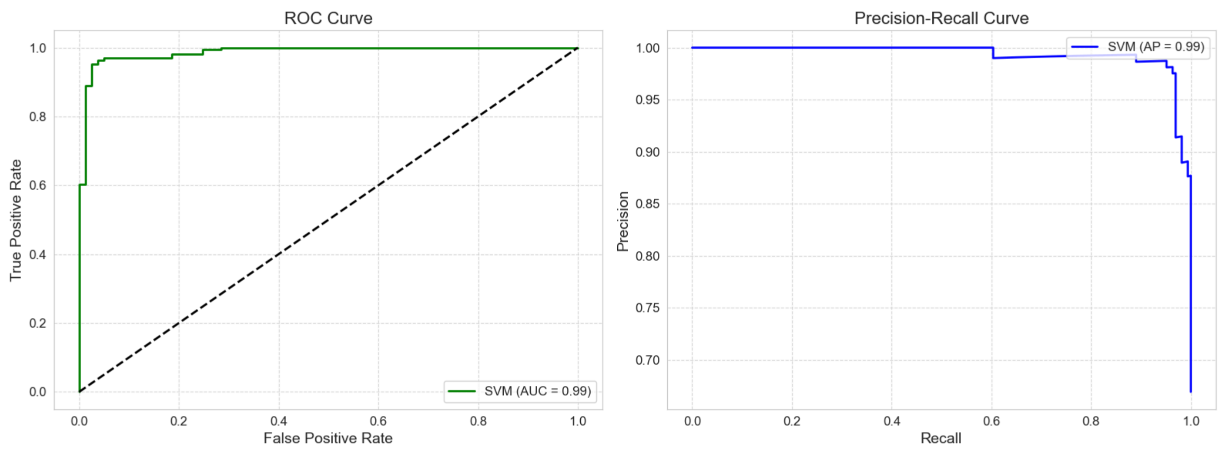
```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, roc_curve, auc, f1_score, recall
import os
import cv2
from sklearn.svm import SVC
import seaborn as sns
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.preprocessing import StandardScaler

# Set seaborn style for better visuals
sns.set_style("whitegrid")
plt.rcParams['font.size'] = 12
plt.rcParams['figure.figsize'] = (10, 6)

# Define tumor classes
path = os.listdir('brain_tumor/Training/')
classes = {'no_tumor': 0, 'pituitary_tumor': 1}

# Load and preprocess data
X = []
Y = []
for cls in classes:
    pth = 'brain_tumor/Training/' + cls
    for j in os.listdir(pth):
        # Load image in grayscale
        img = cv2.imread(pth + '/' + j, 0)
        img = cv2.resize(img, (200, 200))
        img_filtered = cv2.GaussianBlur(img, (5, 5), 0)  # Gaussian blur
        X.append(img_filtered)
        Y.append(classes[cls])

X = np.array(X)
Y = np.array(Y)

# Normalize pixel values
X = X / 255.0

# Reshape images to 1D vectors
X = X.reshape(X.shape[0], -1)

# Split the dataset into training, validation, and testing sets
xtrain, xtest, ytrain, ytest = train_test_split(X, Y, random_state=10, test_
xtrain, xval, ytrain, yval = train_test_split(xtrain, ytrain, random_state=1

# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
xtrain = scaler.fit_transform(xtrain)
xval = scaler.transform(xval)
xtest = scaler.transform(xtest)

# Reshape xtrain back to image format for augmentation
```
Loading [MathJax]/extensions/Safe.js

```python
xtrain_images = xtrain.reshape(xtrain.shape[0], 200, 200, 1)

# Initialize ImageDataGenerator for data augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Fit the data augmentation generator on the training data
datagen.fit(xtrain_images)

# Initialize SVM model with probability=True to enable predict_proba
svm_model = SVC(kernel='rbf', C=1.0, gamma='scale', probability=True, random

# Initialize lists to store accuracy history
train_accuracy_history = []
val_accuracy_history = []
test_accuracy_history = []

# Define the number of steps or subsets of the training data
num_steps = 10
step_size = len(xtrain) // num_steps

# Train the model incrementally and record accuracy
for i in range(1, num_steps + 1):
    subset_size = i * step_size
    xtrain_subset = xtrain[:subset_size]
    ytrain_subset = ytrain[:subset_size]

    # Generate augmented data
    augmented_images, augmented_labels = [], []
    for batch in datagen.flow(xtrain_images[:subset_size], ytrain_subset, ba
        augmented_images.append(batch[0].reshape(batch[0].shape[0], -1))
        augmented_labels.append(batch[1])
        if len(augmented_images) * batch[0].shape[0] >= subset_size:
            break

    # Combine original and augmented data
    xtrain_combined = np.vstack((xtrain_subset, np.vstack(augmented_images))
    ytrain_combined = np.hstack((ytrain_subset, np.hstack(augmented_labels))

    # Standardize the combined training data
    xtrain_combined = scaler.fit_transform(xtrain_combined)

    # Train the model on the combined data
    svm_model.fit(xtrain_combined, ytrain_combined)

    # Record training accuracy
    train_accuracy = svm_model.score(xtrain_combined, ytrain_combined)
    train_accuracy_history.append(train_accuracy)
    # Record validation accuracy
    val_accuracy = svm_model.score(xval, yval)
    val_accuracy_history.append(val_accuracy)

    # Record testing accuracy
    test_accuracy = svm_model.score(xtest, ytest)
    test_accuracy_history.append(test_accuracy)

# Evaluate SVM
svm_train_score = svm_model.score(xtrain, ytrain)
svm_val_score = svm_model.score(xval, yval)
svm_test_score = svm_model.score(xtest, ytest)

# Get predictions from the SVM model
svm_preds = svm_model.predict(xtest)

# Calculate confusion matrix
conf_matrix = confusion_matrix(ytest, svm_preds)

# Extract True Positives (TP), False Positives (FP), True Negatives (TN), ar
true_positives = conf_matrix[1, 1]  # TP
false_positives = conf_matrix[0, 1]  # FP
true_negatives = conf_matrix[0, 0]  # TN
false_negatives = conf_matrix[1, 0]  # FN

# Calculate F1 Score, Recall, and Precision
f1 = f1_score(ytest, svm_preds)
recall = recall_score(ytest, svm_preds)
precision = precision_score(ytest, svm_preds)

# Print the metrics and their formulas
print("\nClassification Metrics:")
print(f"F1 Score: {f1:.4f} (F1 = 2 * (Precision * Recall) / (Precision + Rec
print(f"Recall (Sensitivity): {recall:.4f} (Recall = TP / (TP + FN))")
print(f"Precision: {precision:.4f} (Precision = TP / (TP + FP))")
print(f"Specificity: {true_negatives/(true_negatives+false_positives):.4f} (

# Enhanced Confusion Matrix Visualization
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            annot_kws={'size': 16, 'weight': 'bold'},
            xticklabels=['No Tumor (Predicted)', 'Pituitary (Predicted)'],
            yticklabels=['No Tumor (Actual)', 'Pituitary (Actual)'])
plt.title('Confusion Matrix\nSVM', fontsize=16, pad=20)
plt.xlabel('Predicted Label', fontsize=14, labelpad=15)
plt.ylabel('True Label', fontsize=14, labelpad=15)
plt.xticks(fontsize=12, rotation=45, ha='right')
plt.yticks(fontsize=12, rotation=0)
plt.tight_layout()
plt.show()

# ROC Curve for SVM
svm_probs = svm_model.predict_proba(xtest)[:, 1]
fpr_svm, tpr_svm, _ = roc_curve(ytest, svm_probs)
roc_auc_svm = auc(fpr_svm, tpr_svm)
```

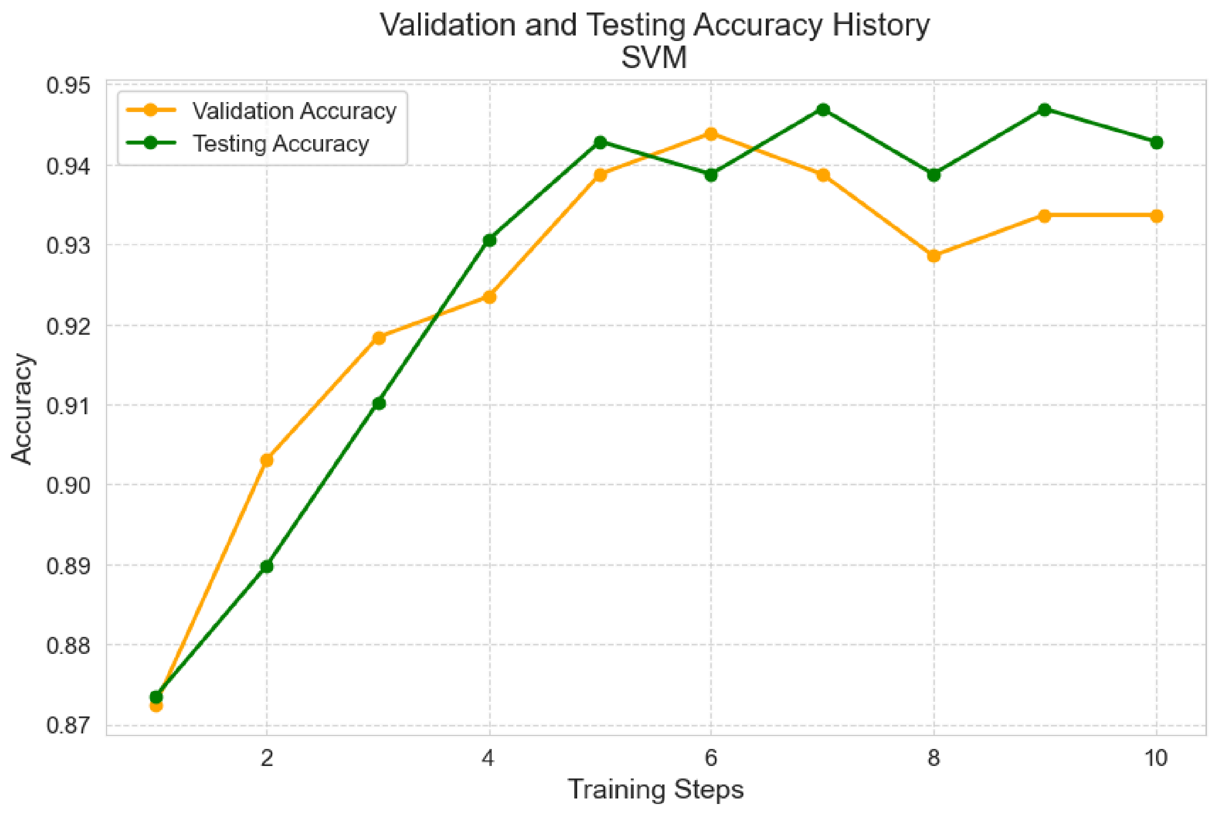
```python
# Precision-Recall Curve
precision_curve, recall_curve, _ = precision_recall_curve(ytest, svm_probs)
average_precision = average_precision_score(ytest, svm_probs)

# Create a figure with two subplots
plt.figure(figsize=(16, 6))

# ROC Curve
plt.subplot(1, 2, 1)
plt.plot(fpr_svm, tpr_svm, label=f'SVM (AUC = {roc_auc_svm:.2f})', color='gr
plt.plot([0, 1], [0, 1], 'k--', linewidth=2)
plt.xlabel('False Positive Rate', fontsize=14)
plt.ylabel('True Positive Rate', fontsize=14)
plt.title('ROC Curve', fontsize=16)
plt.legend(loc='lower right', fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)

# Precision-Recall Curve
plt.subplot(1, 2, 2)
plt.plot(recall_curve, precision_curve, label=f'SVM (AP = {average_precisior
plt.xlabel('Recall', fontsize=14)
plt.ylabel('Precision', fontsize=14)
plt.title('Precision-Recall Curve', fontsize=16)
plt.legend(loc='upper right', fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()

# Plot Accuracy History
plt.figure(figsize=(10, 6))
plt.plot(range(1, num_steps + 1), val_accuracy_history, label='Validation Ac
plt.plot(range(1, num_steps + 1), test_accuracy_history, label='Testing Accu
plt.xlabel('Training Steps', fontsize=14)
plt.ylabel('Accuracy', fontsize=14)
plt.title('Validation and Testing Accuracy History\nSVM', fontsize=16)
plt.legend(fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()

# Print evaluation metrics for SVM
print("\nModel Performance Summary:")
print(f"Validation Accuracy: {svm_val_score:.4f}")
print(f"Testing Accuracy: {svm_test_score:.4f}")
print(f"ROC AUC Score: {roc_auc_svm:.4f}")
print(f"Average Precision: {average_precision:.4f}")
```

```
Classification Metrics:
F1 Score: 0.9578 (F1 = 2 * (Precision * Recall) / (Precision + Recall))
Recall (Sensitivity): 0.9695 (Recall = TP / (TP + FN))
Precision: 0.9464 (Precision = TP / (TP + FP))
Specificity: 0.8889 (Specificity = TN / (TN + FP))
```





Loading [MathJax]/extensions/Safe.js



```
Model Performance Summary:
Validation Accuracy: 0.9337
Testing Accuracy: 0.9429
ROC AUC Score: 0.9872
Average Precision: 0.9934
```

```
In [ ]:
```

Loading [MathJax]/extensions/Safe.js