**Artificial Intelligence**

**Digital Assignment - 1**

**Team Members :**

23BAI1010 – SASANK V

23BAI1054 – AKSHARA GATTUPALLI

23BAI1063 - CHAKKA SAI DHEERAJ

**Task 1 Requirements:**

1. **Implement and evaluate a basic CNN model:**
   * Preprocess the dataset (resizing, normalization, data augmentation).
   * Build a CNN architecture with 3-4 convolutional layers using Conv2D, MaxPooling, Flatten, and Dense layers.
   * Evaluate the model using metrics such as accuracy, F1-score, and AUC-ROC.
2. **Transfer Learning:**
   * Fine-tune pretrained CNN models (ResNet50, VGG16, or MobileNet).
3. **Optimize hyperparameters:**
   * Experiment with learning rates and batch sizes.

**Dataset Preparation**

**Preprocessing Steps:**

* **Resizing:** All images are resized to (224, 224) for compatibility with pretrained models and for uniformity in input dimensions.
* **Data Augmentation:** Applied transformations include:
  + Random horizontal flipping.
  + Random rotation up to 10 degrees.
  + Random affine transformations.
  + Color jitter (brightness and contrast adjustments).
* **Normalization:** Images are normalized with mean [0.485, 0.456, 0.406] and standard deviation [0.229, 0.224, 0.225].

**Basic CNN Architecture**

The basic CNN model consists of four convolutional blocks, followed by a fully connected classifier:

**Convolutional Layers:**

* **First Block:**
  + Conv2D with 32 filters, kernel size 3x3, ReLU activation.
  + Batch normalization and MaxPooling.
* **Second Block:**
  + Conv2D with 64 filters, kernel size 3x3, ReLU activation.
  + Batch normalization and MaxPooling.
* **Third Block:**
  + Conv2D with 128 filters, kernel size 3x3, ReLU activation.
  + Batch normalization and MaxPooling.
* **Fourth Block:**
  + Conv2D with 256 filters, kernel size 3x3, ReLU activation.
  + Batch normalization and MaxPooling.

**Classifier:**

* Flatten layer.
* Fully connected layer with 512 units and ReLU activation.
* Dropout (50%).
* Output layer with 2 units (softmax activation).

**Loss and Optimization:**

* **Loss Function:** CrossEntropyLoss.
* **Optimizer:** Adam with a learning rate of 0.001.
* **Scheduler:** ReduceLROnPlateau to adjust learning rate based on validation accuracy.

**Pretrained Models**

Transfer learning was performed using:

* **ResNet50:** Final fully connected layer modified to output 2 classes.
* **VGG16:** Final classifier layer modified to output 2 classes.
* **MobileNetV2:** Final classifier layer modified to output 2 classes.

**Model Training**

**Data Splitting:**

* The dataset was split into training and validation sets (80:20 ratio) with stratified sampling to maintain class balance.

**Training Details:**

* **Batch Size:** 32.
* **Number of Epochs:** 30.
* **Device:** CUDA (GPU) if available, else CPU.

**Metrics:**

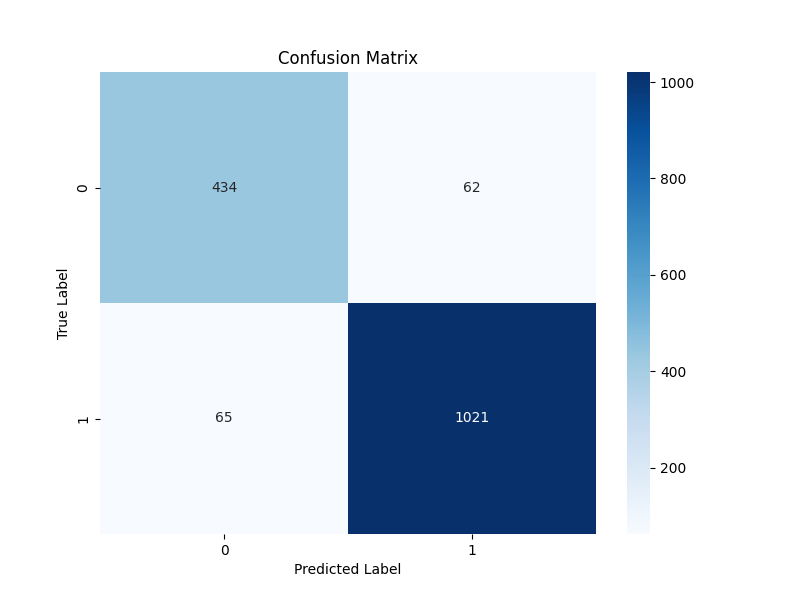
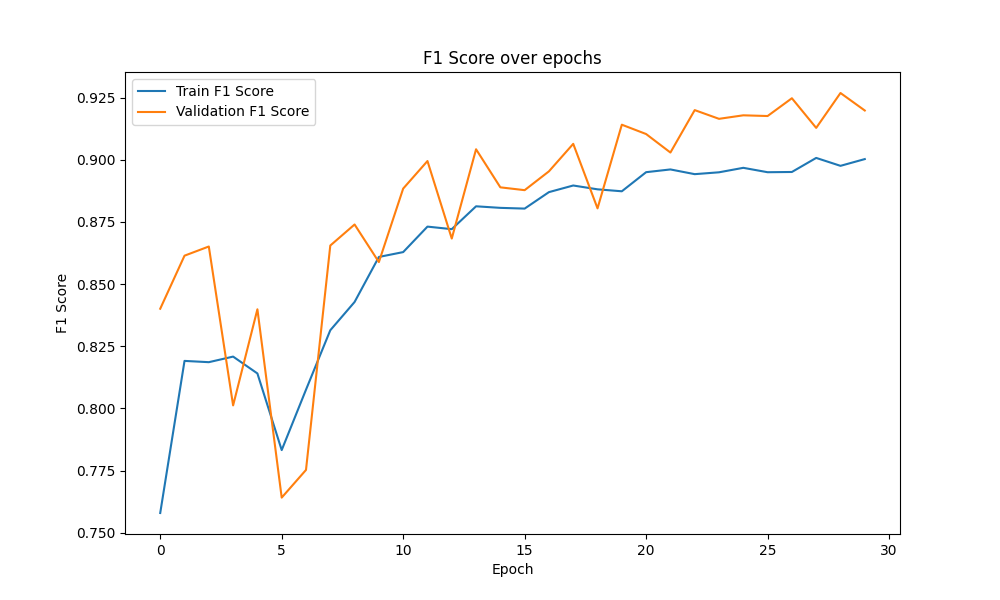
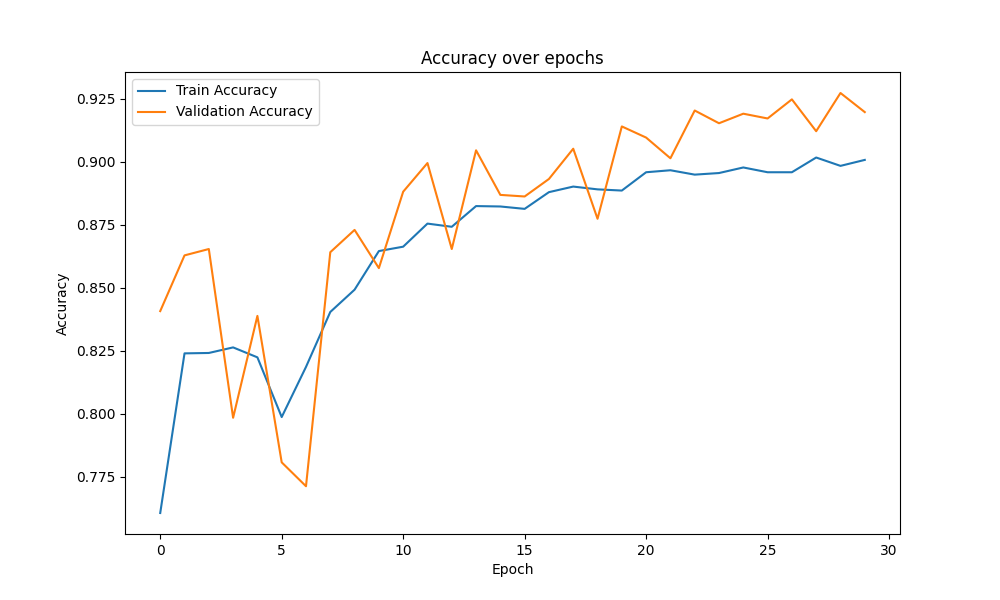
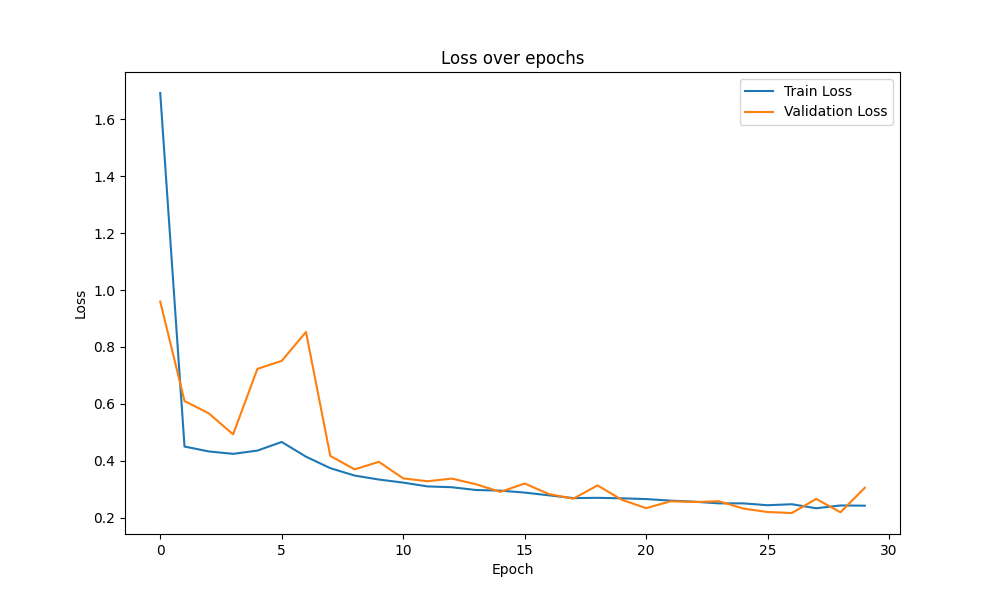
* **Accuracy:** Measures overall classification performance.
* **F1-Score:** Evaluates the balance between precision and recall.
* **AUC-ROC:** Measures the model’s ability to distinguish between classes.

**Results and Visualization**

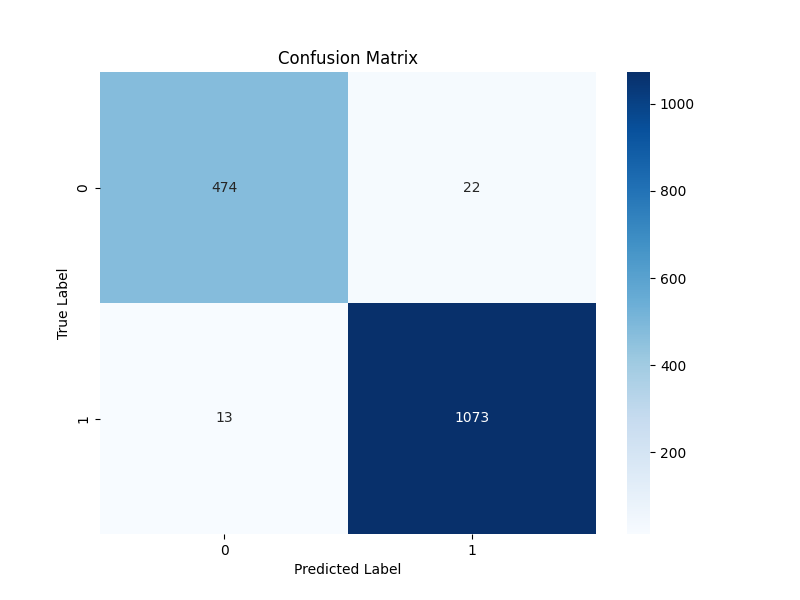
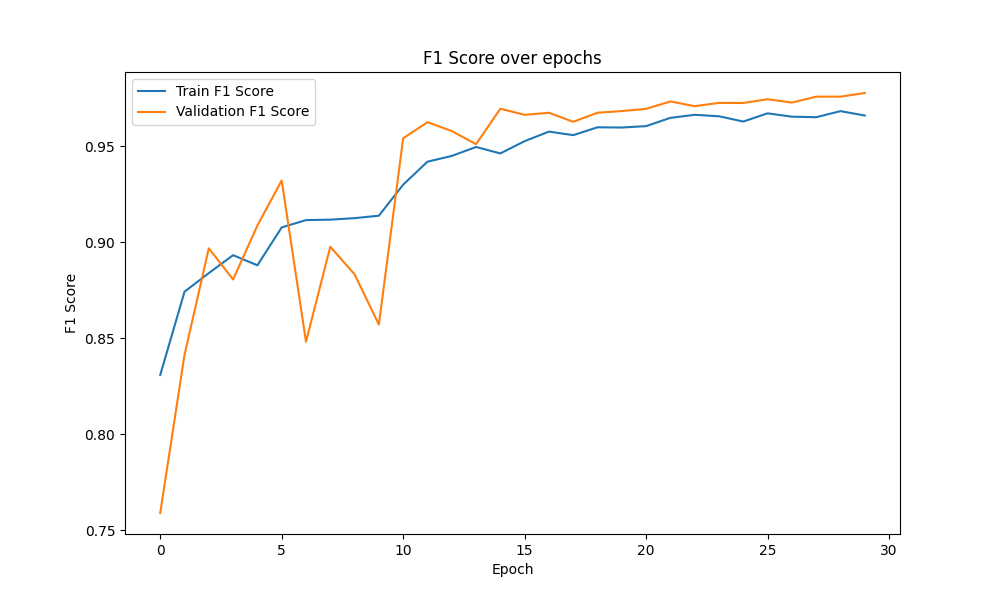
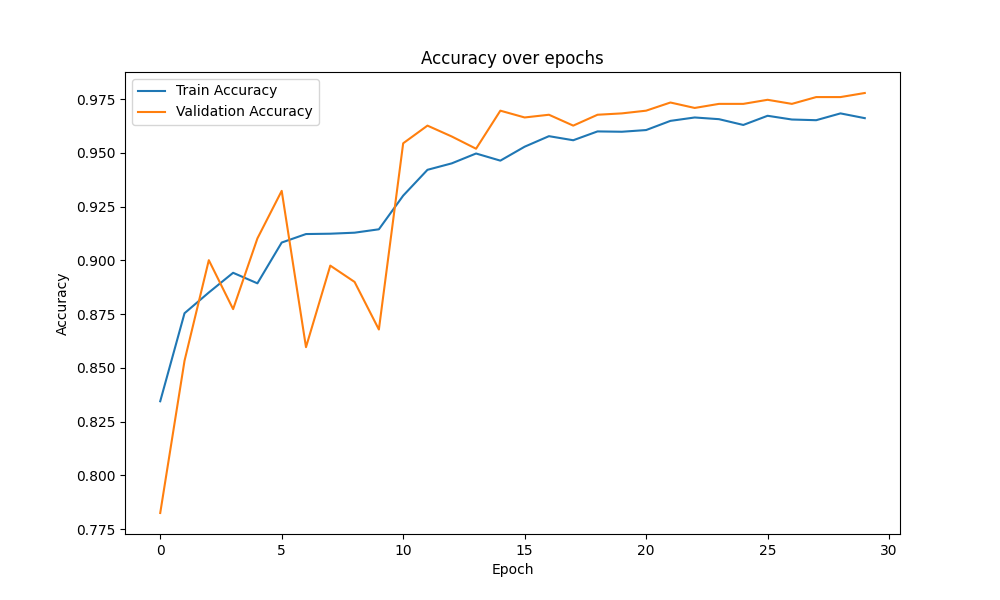
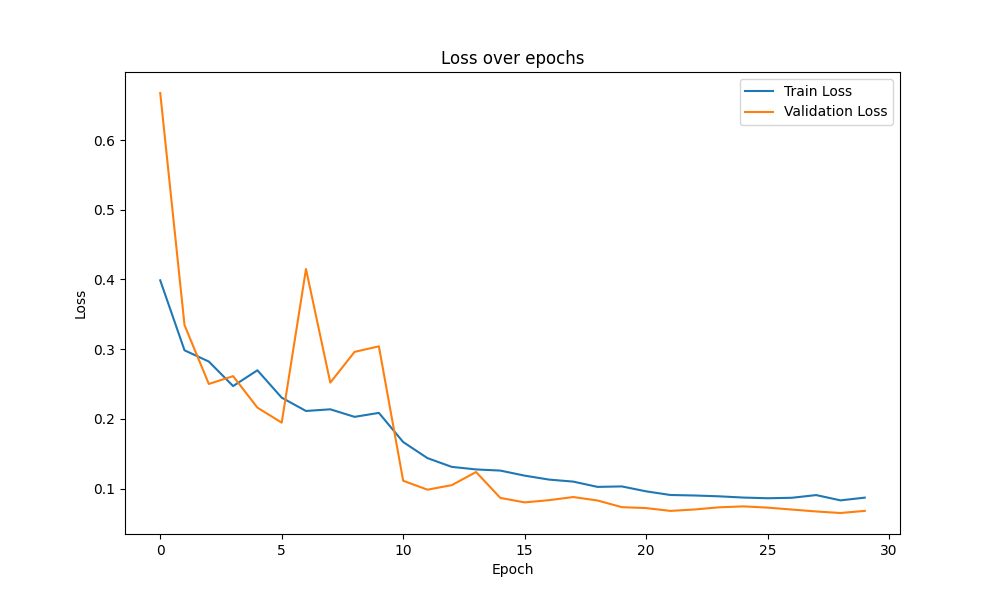
**Training and Validation Metrics:**

* Plots of loss, accuracy, and F1-score over epochs were saved.
* Confusion matrix visualization was generated to understand misclassifications.

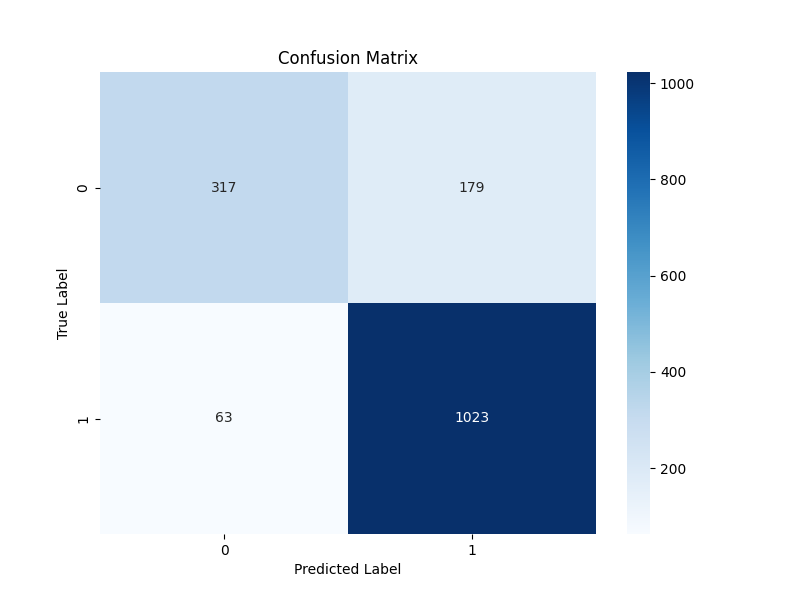
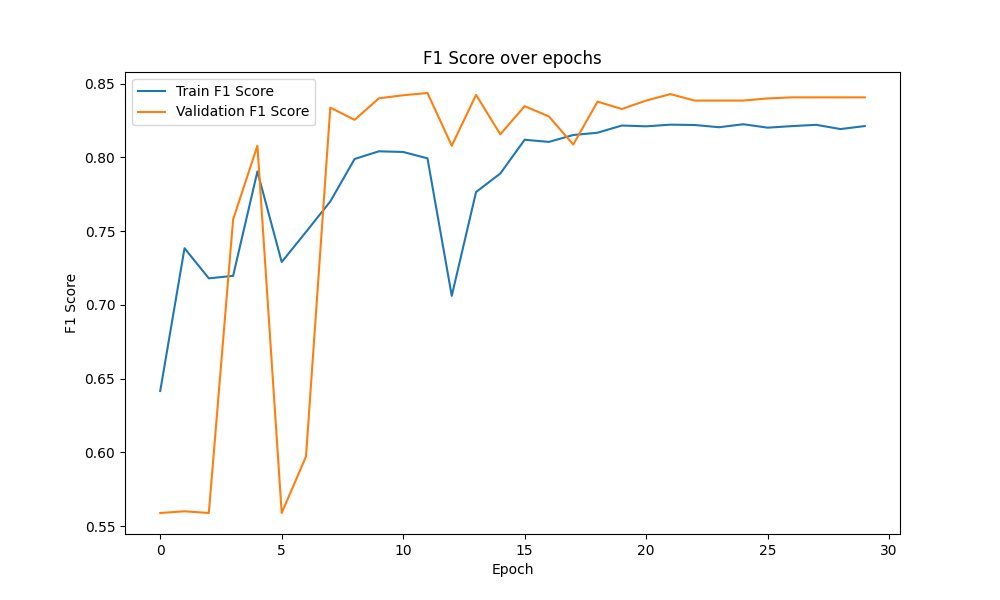
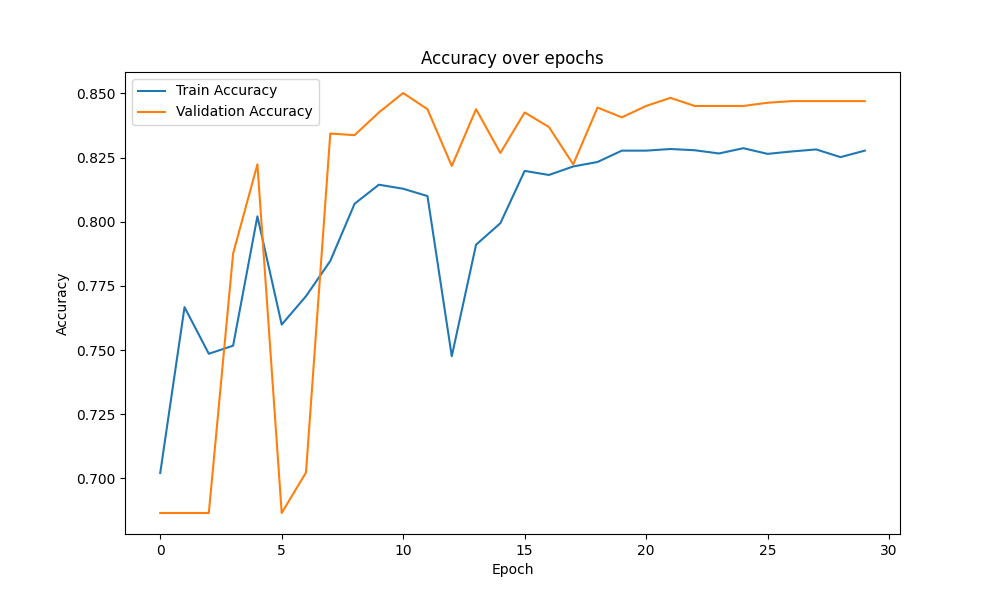
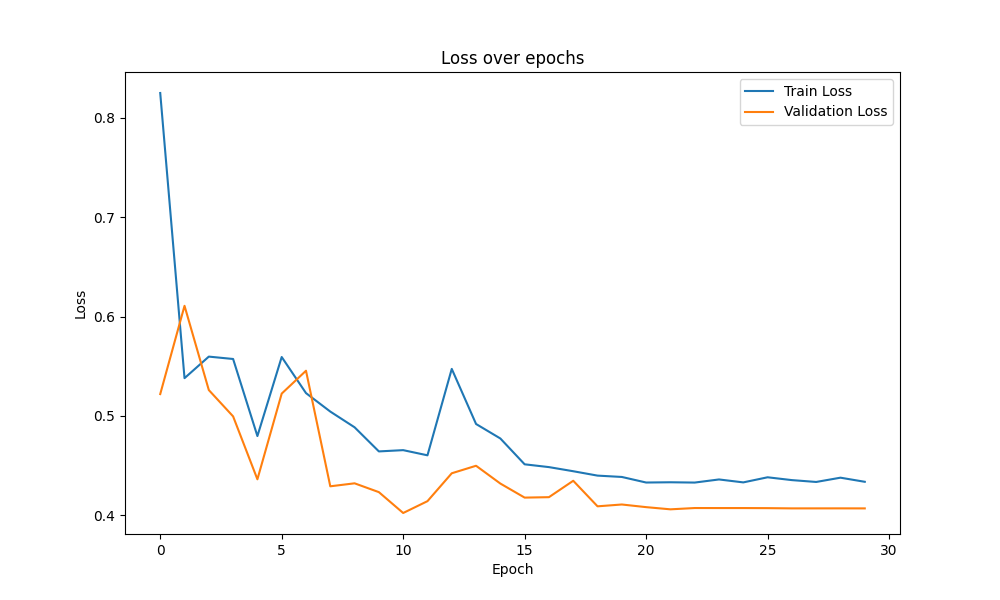
**1.Basic CNN**



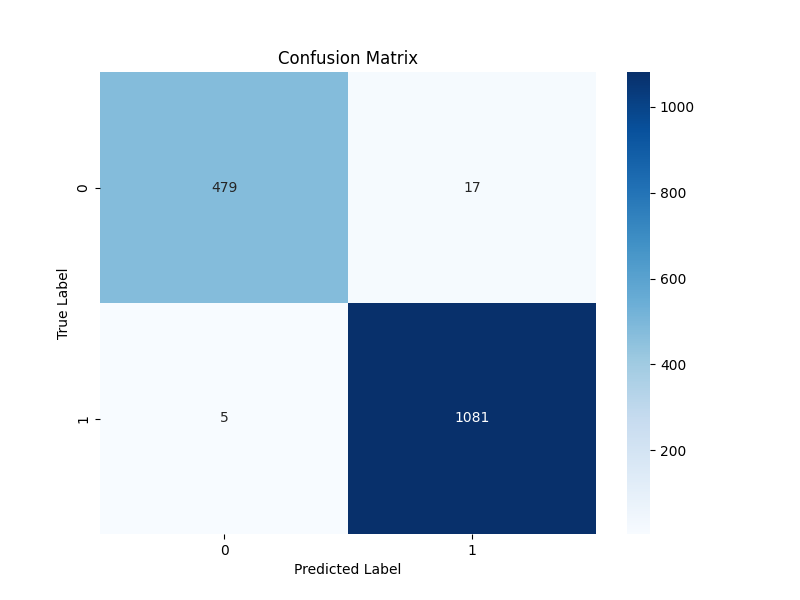
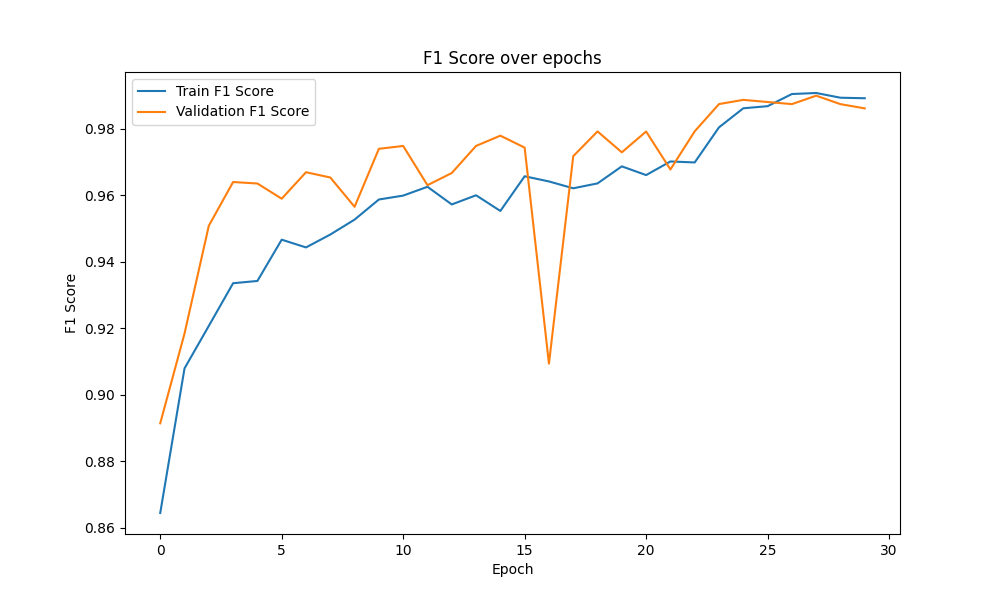
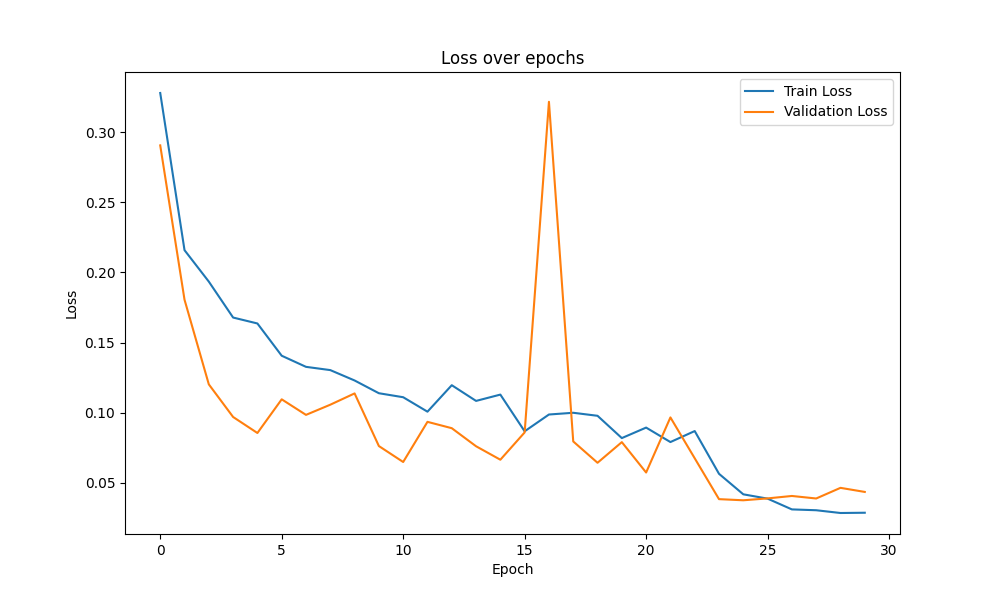
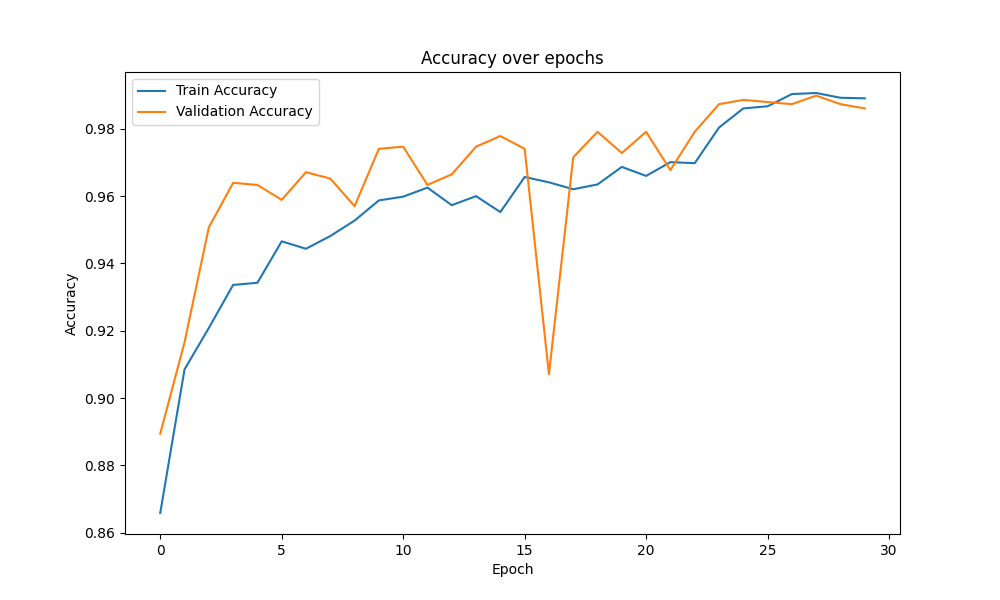
**2.ResNet50**



**3. VGG1**



1. **MobileNetV2**



**Hyperparameter Optimization**

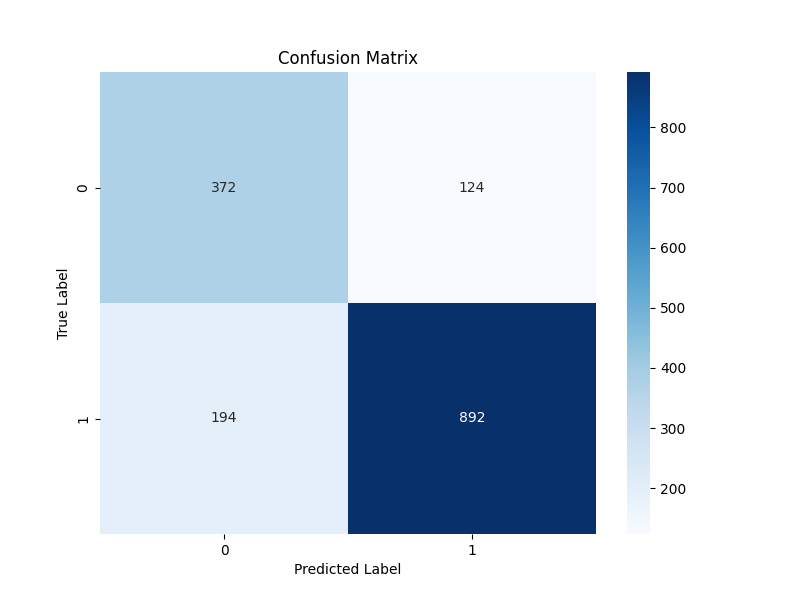
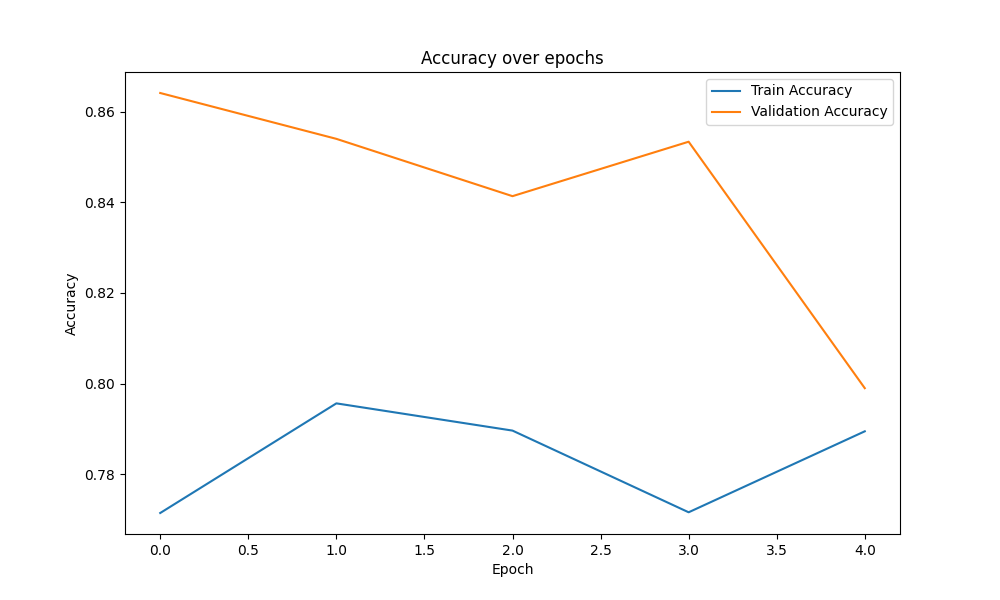
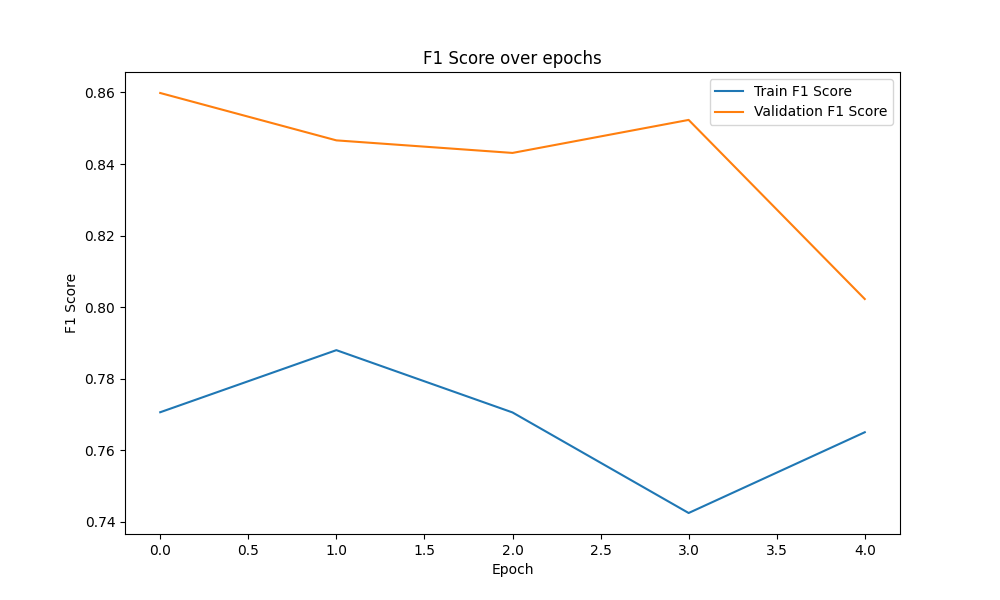
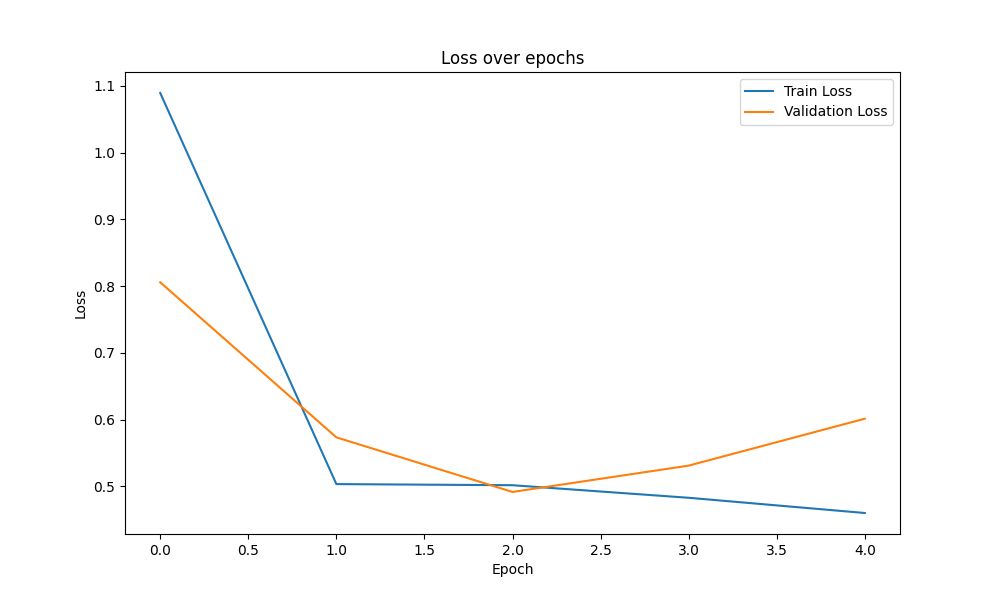
**Parameters Explored:**

* **Learning Rates:** [0.1, 0.01, 0.001].
* **Batch Sizes:** [16, 32, 64].

**Results:**

* Best learning rate: **0.001**.
* Best batch size: **16**.





**Code :**

import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import Dataset, DataLoader

from torchvision import transforms, models

import os

from PIL import Image

import numpy as np

from sklearn.metrics import accuracy\_score, f1\_score, roc\_auc\_score, confusion\_matrix

from sklearn.model\_selection import train\_test\_split

import matplotlib.pyplot as plt

import seaborn as sns

from torch.optim.lr\_scheduler import ReduceLROnPlateau

class BreastCancerDataset(Dataset):

    def \_\_init\_\_(self, image\_paths, labels, transform=None):

        self.image\_paths = image\_paths

        self.labels = labels

        self.transform = transform

    def \_\_len\_\_(self):

        return len(self.image\_paths)

    def \_\_getitem\_\_(self, idx):

        image = Image.open(self.image\_paths[idx]).convert('RGB')

        if self.transform:

            image = self.transform(image)

        label = self.labels[idx]

        return image, label

class BasicCNN(nn.Module):

    def \_\_init\_\_(self):

        super(BasicCNN, self).\_\_init\_\_()

        self.conv\_layers = nn.Sequential(

            # First Conv Block

            nn.Conv2d(3, 32, kernel\_size=3, padding=1),

            nn.ReLU(),

            nn.BatchNorm2d(32),

            nn.MaxPool2d(2),

            # Second Conv Block

            nn.Conv2d(32, 64, kernel\_size=3, padding=1),

            nn.ReLU(),

            nn.BatchNorm2d(64),

            nn.MaxPool2d(2),

            # Third Conv Block

            nn.Conv2d(64, 128, kernel\_size=3, padding=1),

            nn.ReLU(),

            nn.BatchNorm2d(128),

            nn.MaxPool2d(2),

            # Fourth Conv Block

            nn.Conv2d(128, 256, kernel\_size=3, padding=1),

            nn.ReLU(),

            nn.BatchNorm2d(256),

            nn.MaxPool2d(2)

        )

        self.classifier = nn.Sequential(

            nn.Flatten(),

            nn.Linear(256 \* 14 \* 14, 512),

            nn.ReLU(),

            nn.Dropout(0.5),

            nn.Linear(512, 2)

        )

    def forward(self, x):

        x = self.conv\_layers(x)

        x = self.classifier(x)

        return x

def get\_pretrained\_model(model\_name, num\_classes=2):

    if model\_name == 'resnet50':

        model = models.resnet50(pretrained=True)

        num\_features = model.fc.in\_features

        model.fc = nn.Linear(num\_features, num\_classes)

    elif model\_name == 'vgg16':

        model = models.vgg16(pretrained=True)

        num\_features = model.classifier[6].in\_features

        model.classifier[6] = nn.Linear(num\_features, num\_classes)

    elif model\_name == 'mobilenet':

        model = models.mobilenet\_v2(pretrained=True)

        num\_features = model.classifier[1].in\_features

        model.classifier[1] = nn.Linear(num\_features, num\_classes)

    return model

class ModelTrainer:

    def \_\_init\_\_(self, model, device, criterion, optimizer, scheduler=None):

        self.model = model

        self.device = device

        self.criterion = criterion

        self.optimizer = optimizer

        self.scheduler = scheduler

        self.best\_val\_score = 0

    def train\_epoch(self, train\_loader):

        self.model.train()

        running\_loss = 0.0

        predictions = []

        true\_labels = []

        for inputs, labels in train\_loader:

            inputs, labels = inputs.to(self.device), labels.to(self.device)

            self.optimizer.zero\_grad()

            outputs = self.model(inputs)

            loss = self.criterion(outputs, labels)

            loss.backward()

            self.optimizer.step()

            running\_loss += loss.item()

            \_, preds = torch.max(outputs, 1)

            predictions.extend(preds.cpu().numpy())

            true\_labels.extend(labels.cpu().numpy())

        epoch\_loss = running\_loss / len(train\_loader)

        epoch\_acc = accuracy\_score(true\_labels, predictions)

        epoch\_f1 = f1\_score(true\_labels, predictions, average='weighted')

        return epoch\_loss, epoch\_acc, epoch\_f1

    def evaluate(self, val\_loader):

        self.model.eval()

        predictions = []

        true\_labels = []

        val\_loss = 0.0

        with torch.no\_grad():

            for inputs, labels in val\_loader:

                inputs, labels = inputs.to(self.device), labels.to(self.device)

                outputs = self.model(inputs)

                loss = self.criterion(outputs, labels)

                val\_loss += loss.item()

                \_, preds = torch.max(outputs, 1)

                predictions.extend(preds.cpu().numpy())

                true\_labels.extend(labels.cpu().numpy())

        val\_loss = val\_loss / len(val\_loader)

        val\_acc = accuracy\_score(true\_labels, predictions)

        val\_f1 = f1\_score(true\_labels, predictions, average='weighted')

        val\_auc = roc\_auc\_score(true\_labels, predictions)

        return val\_loss, val\_acc, val\_f1, val\_auc, predictions, true\_labels

def plot\_metrics(train\_metrics, val\_metrics, metric\_name):

    plt.figure(figsize=(10, 6))

    plt.plot(train\_metrics, label=f'Train {metric\_name}')

    plt.plot(val\_metrics, label=f'Validation {metric\_name}')

    plt.title(f'{metric\_name} over epochs')

    plt.xlabel('Epoch')

    plt.ylabel(metric\_name)

    plt.legend()

    plt.show()

def plot\_confusion\_matrix(true\_labels, predictions):

    cm = confusion\_matrix(true\_labels, predictions)

    plt.figure(figsize=(8, 6))

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

    plt.title('Confusion Matrix')

    plt.ylabel('True Label')

    plt.xlabel('Predicted Label')

    plt.show()

def train\_and\_evaluate(data\_dir, model\_type='basic\_cnn', num\_epochs=30):

    # Set device

    device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

    print(f"Using device: {device}")

    # Data preparation

    transform\_train = transforms.Compose([

        transforms.Resize((224, 224)),

        transforms.RandomHorizontalFlip(),

        transforms.RandomRotation(10),

        transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),

        transforms.ColorJitter(brightness=0.2, contrast=0.2),

        transforms.ToTensor(),

        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

    ])

    transform\_val = transforms.Compose([

        transforms.Resize((224, 224)),

        transforms.ToTensor(),

        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

    ])

    # Prepare data

    image\_paths = []

    labels = []

    for image\_name in os.listdir(data\_dir):

        if image\_name.endswith('.png'):

            parts = image\_name.split('\_')

            tumor\_class = parts[1]

            image\_paths.append(os.path.join(data\_dir, image\_name))

            labels.append(1 if tumor\_class == 'M' else 0)

    # Split data

    train\_paths, val\_paths, train\_labels, val\_labels = train\_test\_split(

        image\_paths, labels, test\_size=0.2, random\_state=42, stratify=labels

    )

    # Create datasets

    train\_dataset = BreastCancerDataset(train\_paths, train\_labels, transform\_train)

    val\_dataset = BreastCancerDataset(val\_paths, val\_labels, transform\_val)

    # Create dataloaders

    train\_loader = DataLoader(train\_dataset, batch\_size=32, shuffle=True)

    val\_loader = DataLoader(val\_dataset, batch\_size=32)

    # Model selection

    if model\_type == 'basic\_cnn':

        model = BasicCNN()

    else:

        model = get\_pretrained\_model(model\_type)

    model = model.to(device)

    # Training setup

    criterion = nn.CrossEntropyLoss()

    optimizer = optim.Adam(model.parameters(), lr=0.001)

    scheduler = ReduceLROnPlateau(optimizer, mode='max', factor=0.1, patience=3)

    # Initialize trainer

    trainer = ModelTrainer(model, device, criterion, optimizer, scheduler)

    # Training history

    train\_losses, train\_accs, train\_f1s = [], [], []

    val\_losses, val\_accs, val\_f1s, val\_aucs = [], [], [], []

    # Training loop

    for epoch in range(num\_epochs):

        # Train

        train\_loss, train\_acc, train\_f1 = trainer.train\_epoch(train\_loader)

        # Evaluate

        val\_loss, val\_acc, val\_f1, val\_auc, predictions, true\_labels = trainer.evaluate(val\_loader)

        # Update learning rate

        if trainer.scheduler:

            trainer.scheduler.step(val\_acc)

        # Save metrics

        train\_losses.append(train\_loss)

        train\_accs.append(train\_acc)

        train\_f1s.append(train\_f1)

        val\_losses.append(val\_loss)

        val\_accs.append(val\_acc)

        val\_f1s.append(val\_f1)

        val\_aucs.append(val\_auc)

        print(f'Epoch {epoch+1}/{num\_epochs}:')

        print(f'Train Loss: {train\_loss:.4f}, Train Acc: {train\_acc:.4f}, Train F1: {train\_f1:.4f}')

        print(f'Val Loss: {val\_loss:.4f}, Val Acc: {val\_acc:.4f}, Val F1: {val\_f1:.4f}, Val AUC: {val\_auc:.4f}')

        # Save best model

        if val\_acc > trainer.best\_val\_score:

            trainer.best\_val\_score = val\_acc

            torch.save(model.state\_dict(), f'best\_model\_{model\_type}.pth')

    # Plot metrics

    plot\_metrics(train\_losses, val\_losses, 'Loss')

    plot\_metrics(train\_accs, val\_accs, 'Accuracy')

    plot\_metrics(train\_f1s, val\_f1s, 'F1 Score')

    plot\_confusion\_matrix(true\_labels, predictions)

    return model, trainer.best\_val\_score

def hyperparameter\_optimization(data\_dir, model\_type='basic\_cnn'):

    learning\_rates = [0.1, 0.01, 0.001]

    batch\_sizes = [16, 32, 64]

    best\_params = {'lr': None, 'batch\_size': None}

    best\_score = 0

    for lr in learning\_rates:

        for batch\_size in batch\_sizes:

            print(f"\nTrying lr={lr}, batch\_size={batch\_size}")

            \_, val\_score = train\_and\_evaluate(

                data\_dir,

                model\_type=model\_type,

                num\_epochs=5  # Reduced epochs for faster optimization

            )

            if val\_score > best\_score:

                best\_score = val\_score

                best\_params['lr'] = lr

                best\_params['batch\_size'] = batch\_size

    print("\nBest parameters:")

    print(f"Learning rate: {best\_params['lr']}")

    print(f"Batch size: {best\_params['batch\_size']}")

    return best\_params

if \_\_name\_\_ == "\_\_main\_\_":

    data\_dir = "../datasets/all\_images"

    # Train and evaluate basic CNN

    print("Training Basic CNN...")

    train\_and\_evaluate(data\_dir, model\_type='basic\_cnn')

    # Train and evaluate pretrained models

    for model\_type in ['resnet50', 'vgg16', 'mobilenet']:

        print(f"\nTraining {model\_type}...")

        train\_and\_evaluate(data\_dir, model\_type=model\_type)

    # Perform hyperparameter optimization

    print("\nOptimizing hyperparameters...")

    best\_params = hyperparameter\_optimization(data\_dir)