```
!pip install -q gdown pillow matplotlib tensorflow numpy
```

```
import os
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.applications import EfficientNetB0, VGG16, ResNet50
from tensorflow.keras import layers
import gdown

print(f"TensorFlow version: {tf.__version__}")
print(f"GPU Available: {len(tf.config.list_physical_devices('GPU')) > 0}")
```

```
# Model configuration
IMG_SIZE = 224
CLASSES = ['Non Demented', 'Very Mild Demented', 'Mild Demented', 'Moderate Demented']
NUM_CLASSES = len(CLASSES)

# Display configuration
print("="*60)
print("ALZHEIMER'S DETECTION SYSTEM - PRE-TRAINED MODEL")
print("="*60)
print(f"Image Size: {IMG_SIZE}x{IMG_SIZE}")
print(f"Classes: {NUM_CLASSES}")
for i, cls in enumerate(CLASSES):
    print(f"  {i}: {cls}")
print("="*60)
```

## 4️⃣ Download Pre-trained Model

We'll download a pre-trained Alzheimer's detection model from Google Drive.

```
def download_pretrained_model():
    """
    Download pre-trained model from Google Drive.

    Note: This uses a publicly shared model. You can also:
    1. Upload your own trained model
    2. Use models from Kaggle or other sources
    """

    print("📥 Downloading pre-trained model...")

    # Try multiple sources for pre-trained models
    model_sources = [
        {
            'name': 'Alzheimer Detection Model (EfficientNetB0)',
            'url': 'https://github.com/smaranjitghose/AlzheimerNet/releases/download/v1.0/alzheimer_model.h5',
            'filename': 'alzheimer_pretrained.h5'
        }
    ]

    # Try to download from available sources
    for source in model_sources:
        try:
            print(f"\nTrying to download: {source['name']}")

            # Download using gdown or wget
            if 'drive.google.com' in source['url']:
                gdown.download(source['url'], source['filename'], quiet=False)
            else:
                !wget -O {source['filename']} {source['url']}

            if os.path.exists(source['filename']):
                print(f"✅ Successfully downloaded: {source['filename']}")
                return source['filename']
        except Exception as e:
            print(f"⚠️ Failed to download from this source: {e}")
            continue

    print("\n❌ Could not download pre-trained model from available sources.")
    print("\n📝 Alternative options:")
    print("1. Upload your own trained model using the cell below")
    print("2. Use a simple pre-built model (will create one for you)")
```

```python
    return None

# Try to download
model_path = download_pretrained_model()
```

```python
from google.colab import files

print("📤 Upload your pre-trained model file (.h5 or .keras):")
print("\nIf you don't have one, skip this cell and use Option B below.")
print("\nYou can get pre-trained models from:")
print("- Kaggle: https://www.kaggle.com/models")
print("- GitHub repositories with Alzheimer's detection models")
print("- Your own previously trained models")

uploaded = files.upload()

if uploaded:
    model_path = list(uploaded.keys())[0]
    print(f"\n✅ Uploaded model: {model_path}")
else:
    print("\nNo file uploaded. Will use Option B.")
```

```python
def build_pretrained_model(architecture='efficientnet'):
    """
    Build a model using pre-trained weights from ImageNet.
    This provides a good starting point even without Alzheimer-specific training.
    """
    print(f"\n🏗️ Building model with {architecture} architecture...")

    # Choose base model
    if architecture == 'efficientnet':
        base_model = EfficientNetB0(
            include_top=False,
            weights='imagenet',
            input_shape=(IMG_SIZE, IMG_SIZE, 3)
        )
    elif architecture == 'resnet':
        base_model = ResNet50(
            include_top=False,
            weights='imagenet',
            input_shape=(IMG_SIZE, IMG_SIZE, 3)
        )
    else:  # vgg16
        base_model = VGG16(
            include_top=False,
            weights='imagenet',
            input_shape=(IMG_SIZE, IMG_SIZE, 3)
        )

    base_model.trainable = False

    # Build complete model
    inputs = keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
    x = base_model(inputs, training=False)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.BatchNormalization()(x)
    x = layers.Dense(256, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.3)(x)
    outputs = layers.Dense(NUM_CLASSES, activation='softmax')(x)

    model = keras.Model(inputs, outputs)

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

    print("✅ Model built successfully!")
    print(f"   Total parameters: {model.count_params():,}")

    return model

# If no model is available, create one
if model_path is None or not os.path.exists(model_path):
    print("\n⚠️ No pre-trained model found.")
    print("Creating a model with ImageNet pre-trained weights...")
    print("\nNote: This model uses ImageNet features which are general-purpose.")
    print("For best results, use a model specifically trained on Alzheimer's MRI data.")

    model = build_pretrained_model('efficientnet')
    model_loaded = True
else:
    model_loaded = False
```

## 7️⃣ Load the Model

```python
if not model_loaded and model_path and os.path.exists(model_path):
    print(f"📁 Loading model from: {model_path}")

    try:
        model = keras.models.load_model(model_path)
        print("✅ Model loaded successfully!")
        print(f"   Total parameters: {model.count_params():,}")
        model_loaded = True
    except Exception as e:
        print(f"❌ Error loading model: {e}")
        print("\nTrying alternative loading method...")
        try:
            model = keras.models.load_model(model_path, compile=False)
            model.compile(
                optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy']
            )
            print("✅ Model loaded successfully (without compilation)!")
            model_loaded = True
        except Exception as e2:
            print(f"❌ Failed to load model: {e2}")
            print("\nCreating a new model instead...")
            model = build_pretrained_model('efficientnet')
            model_loaded = True

# Display model summary
if model_loaded:
    print("\n" + "="*60)
    print("MODEL SUMMARY")
    print("="*60)
    model.summary()
    print("="*60)
```

```python
def preprocess_image(image_path):
    """Preprocess image for prediction"""
    img = Image.open(image_path).convert('RGB')
    img = img.resize((IMG_SIZE, IMG_SIZE))
    img_array = np.array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)
    return img, img_array

def predict_alzheimer(image_path, model, show_plot=True):
    """
    Predict Alzheimer's disease stage from MRI scan
    """
    # Preprocess
    img, img_array = preprocess_image(image_path)

    # Predict
    predictions = model.predict(img_array, verbose=0)
    predicted_class_idx = np.argmax(predictions[0])
    predicted_class = CLASSES[predicted_class_idx]
    confidence = predictions[0][predicted_class_idx] * 100

    # Visualize if requested
    if show_plot:
        fig, axes = plt.subplots(1, 2, figsize=(15, 5))

        # Show image
        axes[0].imshow(img)
        axes[0].axis('off')
        axes[0].set_title(
            f'Predicted: {predicted_class}\nConfidence: {confidence:.2f}%',
            fontsize=16, fontweight='bold', pad=20
        )

        # Show probabilities
        colors = ['#2ecc71', '#f39c12', '#e67e22', '#e74c3c']
        bar_colors = [colors[i] if i == predicted_class_idx else '#95a5a6'
                      for i in range(len(CLASSES))]

        bars = axes[1].barh(CLASSES, predictions[0] * 100, color=bar_colors)
        axes[1].set_xlabel('Confidence (%)', fontsize=12)
        axes[1].set_title('Class Probabilities', fontsize=16, fontweight='bold', pad=20)
        axes[1].set_xlim(0, 100)
        axes[1].grid(axis='x', alpha=0.3)

        # Add percentage labels on bars
        for i, (bar, prob) in enumerate(zip(bars, predictions[0])):
            width = bar.get_width()
            axes[1].text(width + 2, bar.get_y() + bar.get_height()/2,
                        f'{prob*100:.1f}%',
                        ha='left', va='center', fontsize=10, fontweight='bold')

        plt.tight_layout()
        plt.show()

    return {
        'predicted_class': predicted_class,
        'confidence': confidence,
        'all_probabilities': {cls: float(prob * 100)
                             for cls, prob in zip(CLASSES, predictions[0])}
    }

def get_severity_info(predicted_class):
    """Get information about the prediction"""
    info = {
        'Non Demented': {
            'emoji': '✅',
            'severity': 'Normal',
            'description': 'No signs of dementia detected.',
            'recommendation': 'Continue with regular health checkups.'
        },
        'Very Mild Demented': {
            'emoji': '⚠️',
            'severity': 'Very Mild',
            'description': 'Early stage cognitive decline detected.',
            'recommendation': 'Consult with a neurologist for evaluation.'
        },
        'Mild Demented': {
            'emoji': '⚠️',
            'severity': 'Mild',
            'description': 'Noticeable cognitive impairment present.',
            'recommendation': 'Medical consultation recommended.'
        },
        'Moderate Demented': {
            'emoji': '🚨',
            'severity': 'Moderate',
            'description': 'Significant cognitive impairment detected.',
            'recommendation': 'Immediate medical attention required.'
        }
    }
    return info.get(predicted_class, {})

print("✅ Helper functions loaded successfully!")
```

```python
# Create a sample directory
!mkdir -p sample_images

print("📥 You can download sample MRI images from:")
print("\n1. Kaggle Dataset:")
print("   https://www.kaggle.com/datasets/tourist55/alzheimers-dataset-4-class-of-images")
print("\n2. Sample Brain MRI images:")
print("   - Google 'brain MRI sample images'")
print("   - Use images from medical databases (with permission)")
print("\n3. Or upload your own MRI scans in the next cell")

# Try to download a sample image
try:
    !wget -q -O sample_images/sample_mri.jpg "https://prod-images-static.radiopaedia.org/images/820/35c0f1c6a9e52a52b5b
    if os.path.exists('sample_images/sample_mri.jpg'):
        print("\n✅ Downloaded a sample MRI image: sample_images/sample_mri.jpg")
except:
    print("\n⚠️ Could not download sample image. Please upload your own.")
```

```python
from google.colab import files

print("📤 Upload your brain MRI image(s):")
print("\nSupported formats: JPG, JPEG, PNG")
print("\nNote: This is for educational purposes only.")
print("Always consult medical professionals for diagnosis.\n")
```

```python

uploaded = files.upload()

uploaded_images = list(uploaded.keys())
print(f"\n✅ Uploaded {len(uploaded_images)} image(s)")
```

```python
# Select image to analyze
if uploaded_images:
    image_to_analyze = uploaded_images[0]
elif os.path.exists('sample_images/sample_mri.jpg'):
    image_to_analyze = 'sample_images/sample_mri.jpg'
else:
    print("❌ No image available. Please upload an image first.")
    image_to_analyze = None

if image_to_analyze:
    print(f"\n🔬 Analyzing: {image_to_analyze}")
    print("="*60)

    # Make prediction
    result = predict_alzheimer(image_to_analyze, model, show_plot=True)

    # Display detailed results
    print("\n" + "="*60)
    print("PREDICTION RESULTS")
    print("="*60)

    severity = get_severity_info(result['predicted_class'])

    print(f"\n{severity['emoji']} Predicted Class: {result['predicted_class']}")
    print(f"📊 Confidence: {result['confidence']:.2f}%")
    print(f"\n📝 Description: {severity['description']}")
    print(f"💡 Recommendation: {severity['recommendation']}")

    print("\n" + "="*60)
    print("All Class Probabilities:")
    print("="*60)
    for cls, prob in result['all_probabilities'].items():
        bar = "█" * int(prob / 2)
        print(f"{cls:25s} {prob:6.2f}% {bar}")
    print("="*60)

    print("\n⚠️  IMPORTANT DISCLAIMER:")
    print("This is an AI prediction tool for educational purposes only.")
    print("It is NOT a medical diagnosis. Always consult healthcare professionals.")
```

```python
if len(uploaded_images) > 1:
    print(f"\n🔬 Analyzing {len(uploaded_images)} images...\n")

    results_table = []

    for i, img_path in enumerate(uploaded_images):
        print(f"\nProcessing {i+1}/{len(uploaded_images)}: {img_path}")
        result = predict_alzheimer(img_path, model, show_plot=False)

        results_table.append({
            'Image': img_path,
            'Prediction': result['predicted_class'],
            'Confidence': f"{result['confidence']:.2f}%"
        })

        print(f"  → {result['predicted_class']} ({result['confidence']:.2f}%)")

    # Display summary table
    print("\n" + "="*80)
    print("BATCH ANALYSIS SUMMARY")
    print("="*80)

    import pandas as pd
    df = pd.DataFrame(results_table)
    print(df.to_string(index=False))

    # Statistics
    print("\n" + "="*80)
    print("STATISTICS")
    print("="*80)
    for cls in CLASSES:
        count = sum(1 for r in results_table if r['Prediction'] == cls)
        percentage = (count / len(results_table)) * 100
        print(f"{cls:25s}: {count:2d} ({percentage:5.1f}%)")
    print("="*80)
```

```
else:
    print("\nℹ️ Upload multiple images to enable batch processing.")
```

```
if len(uploaded_images) > 1:
    # Create a grid visualization
    num_images = min(len(uploaded_images), 6)  # Show max 6 images
    cols = 3
    rows = (num_images + cols - 1) // cols

    fig, axes = plt.subplots(rows, cols, figsize=(15, 5*rows))
    axes = axes.flatten() if num_images > 1 else [axes]

    for i in range(num_images):
        img_path = uploaded_images[i]
        img = Image.open(img_path)

        result = predict_alzheimer(img_path, model, show_plot=False)

        axes[i].imshow(img)
        axes[i].axis('off')

        severity = get_severity_info(result['predicted_class'])
        title = f"{severity['emoji']} {result['predicted_class']}\n{result['confidence']:.1f}%"
        axes[i].set_title(title, fontsize=12, fontweight='bold')

    # Hide extra subplots
    for i in range(num_images, len(axes)):
        axes[i].axis('off')

    plt.tight_layout()
    plt.show()
else:
    print("\nℹ️ Upload multiple images to see grid visualization.")
```

```
# Save results to a CSV file
if uploaded_images:
    import pandas as pd
    from datetime import datetime

    all_results = []

    for img_path in uploaded_images:
        result = predict_alzheimer(img_path, model, show_plot=False)
        all_results.append({
            'Timestamp': datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
            'Image': img_path,
            'Prediction': result['predicted_class'],
            'Confidence (%)': f"{result['confidence']:.2f}",
            'Non Demented (%)': f"{result['all_probabilities']['Non Demented']:.2f}",
            'Very Mild (%)': f"{result['all_probabilities']['Very Mild Demented']:.2f}",
            'Mild (%)': f"{result['all_probabilities']['Mild Demented']:.2f}",
            'Moderate (%)': f"{result['all_probabilities']['Moderate Demented']:.2f}"
        })

    df_results = pd.DataFrame(all_results)

    # Save to CSV
    csv_filename = 'alzheimer_predictions.csv'
    df_results.to_csv(csv_filename, index=False)

    print(f"\n✅ Results saved to: {csv_filename}")
    print("\nPreview:")
    print(df_results.to_string(index=False))

    # Download the CSV
    print("\n📥 Downloading results file...")
    files.download(csv_filename)
```

```
print("="*70)
print("MODEL INFORMATION")
print("="*70)
print(f"\nArchitecture: Transfer Learning with Pre-trained CNN")
print(f"Input Size: {IMG_SIZE}x{IMG_SIZE}x3")
print(f"Number of Classes: {NUM_CLASSES}")
print(f"Total Parameters: {model.count_params():,}")

trainable_params = sum([tf.size(w).numpy() for w in model.trainable_weights])
non_trainable_params = sum([tf.size(w).numpy() for w in model.non_trainable_weights])

print(f"Trainable Parameters: {trainable_params:,}")
print(f"Non-trainable Parameters: {non_trainable_params:,}")

print("\nClasses:")
for i, cls in enumerate(CLASSES):
    print(f"  {i}: {cls}")

print("\n" + "="*70)
print("IMPORTANT NOTES")
print("="*70)
print("""
⚠️  MEDICAL DISCLAIMER:
- This is an AI tool for educational and research purposes ONLY
- NOT approved for clinical diagnosis
- NOT a replacement for professional medical evaluation
- False positives and negatives are possible
- Always consult qualified healthcare professionals

🔬 MODEL LIMITATIONS:
- Performance depends on training data quality
- May not generalize to all MRI types/protocols
- Best used as a screening/triage tool
- Should be validated in clinical settings before any use

📊 For Best Results:
- Use high-quality MRI scans
- Ensure proper image orientation
- Use images similar to training data
- Consider multiple predictions if uncertain
""")
print("="*70)
```

model is trained last ke 3 cell run karna hai

```
!pip install -q streamlit transformers torch pillow
!npm install localtunnel
```

```
⠙⠹⠸⠼⠴
up to date, audited 23 packages in 715ms
⠴
⠴3 packages are looking for funding
⠴  run `npm fund` for details
⠴
2 high severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
⠴
```

```
%%writefile app.py
import streamlit as st
from transformers import pipeline
from PIL import Image

# 1. Load the Pretrained Model from Hugging Face
# We use a Vision Transformer (ViT) fine-tuned on Alzheimer's MRI data
@st.cache_resource
def load_model():
    # This downloads the model ~300MB once
    pipe = pipeline("image-classification", model="dheiver/Alzheimer-MRI-ViT")
    return pipe

model_pipeline = load_model()

# 2. UI Layout
st.title("🧠 Alzheimer's Detection (Pretrained)")
st.markdown("### Model: Vision Transformer (ViT)")
st.write("This app uses a model pre-trained by researchers, so no training is required here.")

file = st.file_uploader("Upload an MRI Scan (JPG/PNG)", type=["jpg", "png", "jpeg"])

if file is not None:
    image = Image.open(file)
    st.image(image, caption="Uploaded Scan", use_column_width=True)

    if st.button("Analyze Scan"):
        with st.spinner('Downloading model & Analyzing...'):
            # The pipeline handles resizing and preprocessing automatically
            results = model_pipeline(image)

            # Results come back as a list of dicts: [{'label': 'NonDemented', 'score': 0.99}, ...]
            top_result = results[0]
            label = top_result['label']
            score = top_result['score'] * 100

            st.success(f"**Diagnosis:** {label}")
            st.info(f"Confidence: {score:.2f}%")

            # Show other probabilities
            st.write("---")
            st.write("**Full Analysis:**")
            for res in results:
                st.write(f"- {res['label']}: {res['score']*100:.1f}%")
```

```
Overwriting app.py
```

```
print("1. COPY THIS IP ADDRESS for the password:")
!wget -q -O - ipv4.icanhazip.com
print("\n2. Click the link below and paste the IP:")
!streamlit run app.py & npx localtunnel --port 8501
```

```
1. COPY THIS IP ADDRESS for the password:
136.116.2.58

2. Click the link below and paste the IP:
⠙
Collecting usage statistics. To deactivate, set browser.gatherUsageStats to false.

your url is: https://huge-nights-smell.loca.lt

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://172.28.0.12:8501
  External URL: http://136.116.2.58:8501

  Stopping...
^C
```