Research report

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

BCSE306L - Artificial Intelligence

DIGITAL ASSIGNMENT 2

By: Yana Jain & Tushar Kumar

Reg: 23BAI1292 & 23BAI1121

BreakHis Binary Classification

Basic Swin Transformer Model

Completed on
Mar 8, 2025

Prepared by
Yana Jain
Tushar Kumar

Key insights

- Dataset Processing
- Evaluation of Model
- Comparing with CNN model

Introduction

Swin Transformer (Shifted Window Transformer)

The **Swin Transformer** is a cutting-edge vision transformer developed to enhance image analysis while addressing the computational limitations of traditional **Vision Transformers (ViT)**. Unlike ViT, which applies self-attention across the entire image, Swin Transformer introduces a **localized window-based attention mechanism** and a **hierarchical structure**, making it more scalable and effective for processing high-resolution images.

Key Features of Swin Transformer

- **Hierarchical Feature Learning:**
  Unlike ViT, which processes images using fixed-size patches, Swin Transformer employs a **multi-scale hierarchical approach**, much like CNNs. This enables it to capture both **fine details** and **broad contextual information**, enhancing its performance in complex image classification tasks.

- **Shifted Window Attention:**
  Traditional transformers compute self-attention across the entire image, leading to high computational costs. Swin Transformer **divides images into small, non-overlapping windows**, applying self-attention within each. To facilitate information exchange between windows, they are **shifted between layers**, allowing the model to capture **long-range dependencies** while maintaining efficiency.

- **Improved Computational Efficiency:**
  By restricting self-attention to **localized windows**, Swin Transformer significantly reduces computational complexity compared to standard transformers. This makes it particularly well-suited for high-resolution images, as it processes smaller regions first before expanding to extract **global features**.

- **Scalability and Adaptability:**
  Due to its layered architecture, Swin Transformer can be applied to a wide range of vision tasks, including **image classification, object detection, and medical imaging**. Its ability to adapt to **different patch sizes and window configurations** makes it more flexible than conventional transformers.

By combining **localized attention with hierarchical global feature learning**, the **Swin Transformer** achieves a **balance between accuracy and computational efficiency**, making it a strong alternative to **CNNs and traditional ViTs** for vision-related applications.

> Both breast tumors benign and malignant can be sorted into different types based on the way the tumoral cells look under the microscope.

Dataset Processing

Why is processing required?

- To ensure compatibility with the model
- To improve model performance
- To handle imbalanced data
- To speed up training

SOURCE CODE

1. sortimages.py

```
import os
import shutil
import random

# Define source and destination paths
SOURCE_PATH = "/home/yanajain/Downloads/BreaKHis_v1
(2)/histology_slides/breast"
DEST_PATH = "/home/yanajain/Downloads/BreaKHis_Processed"

# Train-validation split ratio
TRAIN_SPLIT = 0.8

# Magnification levels
```

```
MAGNIFICATIONS = ["40X", "100X", "200X", "400X"]

# Categories in dataset
CATEGORIES = ["benign", "malignant"]

# Create train/val directories
for split in ["train", "val"]:
	for category in CATEGORIES:
	for mag in MAGNIFICATIONS:
	os.makedirs(os.path.join(DEST_PATH, split, category, mag),
exist_ok=True)

# Function to find images recursively
def get_all_images(base_path, mag_level):
	"""Finds all images in subdirectories containing a specific
magnification level."""
	image_paths = []
	for root, dirs, files in os.walk(base_path):
	if mag_level in root:  # Check if magnification level is in the path
	for file in files:
		if file.endswith((".png", ".jpg", ".jpeg")):
		image_paths.append(os.path.join(root, file))
	return image_paths

# Process each category (benign/malignant)
for category in CATEGORIES:
	category_path = os.path.join(SOURCE_PATH, category)

	# Iterate through magnification levels
	for mag in MAGNIFICATIONS:
	images = get_all_images(category_path, mag)

	# Shuffle images for randomness
	random.shuffle(images)

	# Split into train and validation sets
	split_index = int(len(images) * TRAIN_SPLIT)
	train_images = images[:split_index]
	val_images = images[split_index:]

	# Move images to respective directories
	for img_path in train_images:
	shutil.copy(img_path, os.path.join(DEST_PATH, "train", category,
mag))
```

```
        for img_path in val_images:
            shutil.copy(img_path, os.path.join(DEST_PATH, "val", category, mag))

print("Dataset successfully reorganized!")
```

Here we set 80% dataset for training and 20% dataset for validation

> Before processing the data make sure you remember to set up your environment to run the python code.

Steps to execute

- Step 1: Install python 3.11, since 3.12 does not support Tensorflow
- Step 2: In your ubuntu terminal, create a virtual environment and install the required libraries.
- Step 3: In a python script (sortimages.py), organize them according to different magnification levels.
- Step 4: Run for different magnification levels to find the validation accuracy.
- Step 5: As mentioned, consider various window and patch sizes to process the data.

> **Data augmentation** is a technique that artificially increases the amount of data used to train deep learning models.

```
yanajain@yanajain-IdeaPad-3-15IIL05:~$ python3.11 --version
Python 3.11.11
```

Commands to install python 3.11

sudo apt update && sudo apt upgrade

sudo apt install software-properties-common

sudo add-apt-repository ppa:deadsnakes/ppa

sudo apt update

sudo apt install python3.11

```
yanajain@yanajain-IdeaPad-3-15IIL05:~$ python3.11 --version
Python 3.11.11
yanajain@yanajain-IdeaPad-3-15IIL05:~$ sudo apt install python3.11-venv
[sudo] password for yanajain:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libllvm17t64 python3-netifaces
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  python3.11-distutils python3.11-lib2to3
The following NEW packages will be installed:
  python3.11-distutils python3.11-lib2to3 python3.11-venv
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 3,238 kB of archives.
After this operation, 4,461 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu noble/main amd64 python3.11-lib2to3 all 3.11.11-1+noble1 [82.5 kB]
Get:2 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu noble/main amd64 python3.11-distutils all 3.11.11-1+noble1 [138 kB]
Get:3 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu noble/main amd64 python3.11-venv amd64 3.11.11-1+noble1 [3,018 kB]
Fetched 3,238 kB in 28s (114 kB/s)
Selecting previously unselected package python3.11-lib2to3.
(Reading database ... 193134 files and directories currently installed.)
Preparing to unpack .../python3.11-lib2to3_3.11.11-1+noble1_all.deb ...
Unpacking python3.11-lib2to3 (3.11.11-1+noble1) ...
Selecting previously unselected package python3.11-distutils.
Preparing to unpack .../python3.11-distutils_3.11.11-1+noble1_all.deb ...
Unpacking python3.11-distutils (3.11.11-1+noble1) ...
Selecting previously unselected package python3.11-venv.
Preparing to unpack .../python3.11-venv_3.11.11-1+noble1_amd64.deb ...
Unpacking python3.11-venv (3.11.11-1+noble1) ...
Setting up python3.11-lib2to3 (3.11.11-1+noble1) ...
Setting up python3.11-distutils (3.11.11-1+noble1) ...
Setting up python3.11-venv (3.11.11-1+noble1) ...
yanajain@yanajain-IdeaPad-3-15IIL05:~$ python3.11 -m venv swin_env
yanajain@yanajain-IdeaPad-3-15IIL05:~$ source swin_env/bin/activate
```

```
(swin_env) yanajain@yanajain-IdeaPad-3-15IIL05:~$ pip install tensorflow-addons
Collecting tensorflow-addons
  Downloading tensorflow_addons-0.23.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (1.8 kB)
Requirement already satisfied: packaging in ./swin_env/lib/python3.11/site-packages (from tensorflow-addons) (24.2)
Collecting typeguard<3.0.0,>=2.7 (from tensorflow-addons)
  Downloading typeguard-2.13.3-py3-none-any.whl.metadata (3.6 kB)
Downloading tensorflow_addons-0.23.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (611 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 611.8/611.8 kB 1.1 MB/s eta 0:00:00
Downloading typeguard-2.13.3-py3-none-any.whl (17 kB)
Installing collected packages: typeguard, tensorflow-addons
Successfully installed tensorflow-addons-0.23.0 typeguard-2.13.3
(swin_env) yanajain@yanajain-IdeaPad-3-15IIL05:~$ pip install transformers
Collecting transformers
  Using cached transformers-4.49.0-py3-none-any.whl.metadata (44 kB)
Collecting filelock (from transformers)
  Using cached filelock-3.17.0-py3-none-any.whl.metadata (2.9 kB)
Collecting huggingface-hub<1.0,>=0.26.0 (from transformers)
  Downloading huggingface_hub-0.29.2-py3-none-any.whl.metadata (13 kB)
Requirement already satisfied: numpy>=1.17 in ./swin_env/lib/python3.11/site-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in ./swin_env/lib/python3.11/site-packages (from transformers) (24.2)
Collecting pyyaml>=5.1 (from transformers)
  Downloading PyYAML-6.0.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (2.1 kB)
Collecting regex!=2019.12.17 (from transformers)
  Downloading regex-2024.11.6-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (40 kB)
Requirement already satisfied: requests in ./swin_env/lib/python3.11/site-packages (from transformers) (2.32.3)
Collecting tokenizers<0.22,>=0.21 (from transformers)
  Using cached tokenizers-0.21.0-cp39-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (6.7 kB)
Collecting safetensors>=0.4.1 (from transformers)
  Using cached safetensors-0.5.3-cp38-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.8 kB)
Collecting tqdm>=4.27 (from transformers)
  Using cached tqdm-4.67.1-py3-none-any.whl.metadata (57 kB)
Collecting fsspec>=2023.5.0 (from huggingface-hub<1.0,>=0.26.0->transformers)
  Using cached fsspec-2025.2.0-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: typing-extensions>=3.7.4.3 in ./swin_env/lib/python3.11/site-packages (from huggingface-hub<1.0,>=0.26.0->transformers) (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in ./swin_env/lib/python3.11/site-packages (from requests->transformers) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in ./swin_env/lib/python3.11/site-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in ./swin_env/lib/python3.11/site-packages (from requests->transformers) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in ./swin_env/lib/python3.11/site-packages (from requests->transformers) (2025.1.31)
Using cached transformers-4.49.0-py3-none-any.whl (10.0 MB)
Downloading huggingface_hub-0.29.2-py3-none-any.whl (468 kB)
Downloading PyYAML-6.0.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (762 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 763.0/763.0 kB 585.9 kB/s eta 0:00:00
Downloading regex-2024.11.6-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (792 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 792.7/792.7 kB 539.5 kB/s eta 0:00:00
Using cached safetensors-0.5.3-cp38-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (471 kB)
```

TIMM (Torch Image Models) is an advanced PyTorch library that provides a wide range of pre-trained deep learning models for image classification, object detection, and segmentation. It includes optimized versions of architectures like Swin Transformer, ViTs, and CNNs, facilitating efficient training and fine-tuning on personalized datasets. TIMM is built for versatility, scalability, and high-performance computing, making it well-suited for applications such as breast cancer classification, where accuracy and computational efficiency are paramount. Its integrated optimizations streamline model loading, preprocessing, and evaluation, enabling easy comparison of various architectures within a consolidated framework.

Evaluating the Model

Impact of Patch and Window Sizes in Swin Transformer

1. Model Performance (Accuracy, Loss)

- Smaller patch sizes (32) tend to retain more fine-grained information, leading to better feature extraction and classification accuracy.
- Larger patch sizes (64) reduce spatial resolution, which might lead to lower accuracy for detailed tasks like medical image classification.
- Larger window sizes (128) allow the model to capture more global context but increase computation time.
- Smaller window sizes (64) focus more on local features and might miss long-range dependencies.

2. Computational Efficiency (Memory, Speed)

- Patch Size 32 → Higher memory usage & slower inference due to more patches
- Patch Size 64 → Lower memory usage & faster inference but might lose fine details
- Window Size 64 → Faster computation with more localized feature extraction
- Window Size 128 → More expensive but better global representation

Tuning **patch and window sizes** during **model evaluation** directly influences **efficiency, accuracy, and feature extraction quality.** Swin Transformer achieves a **balance between computational cost and accuracy**, making it highly effective for **breast cancer histopathology analysis**.

> We applied patch sizes of 32, 64 and window sizes of 64, 128 on all different magnification levels (40x, 100x, 200x, 400x).

SOURCE CODE

swin.py

```
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define the magnifications and their respective directories
magnifications = ['40X', '100X', '200X', '400X']
base_dir = '/home/yanajain/Downloads/BreaKHis_Processed'
train_dirs = {mag: [os.path.join(base_dir, 'train', 'benign', mag),
                    os.path.join(base_dir, 'train', 'malignant', mag)] for mag in magnifications}
val_dirs = {mag: [os.path.join(base_dir, 'val', 'benign', mag),
                  os.path.join(base_dir, 'val', 'malignant', mag)] for mag in magnifications}

# Function to create and compile the model
def create_model(input_shape):
```

```
    model = tf.keras.models.Sequential([
    tf.keras.Input(shape=input_shape),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam',
        loss='binary_crossentropy',
        metrics=['accuracy'])
    return model

# Helper function to create generators
def create_generators(magnification, train_or_val, target_size):
    benign_dir = os.path.join(base_dir, train_or_val, 'benign', magnification)
    malignant_dir = os.path.join(base_dir, train_or_val, 'malignant', magnification)

    datagen = ImageDataGenerator(rescale=1./255,
            rotation_range=20,
            width_shift_range=0.2,
            height_shift_range=0.2,
            shear_range=0.2,
            zoom_range=0.2,
            horizontal_flip=True if train_or_val == 'train' else False)

    combined_dir = os.path.join(base_dir, train_or_val, magnification)
    os.makedirs(combined_dir, exist_ok=True)

    generator = datagen.flow_from_directory(combined_dir,
                target_size=target_size,
                batch_size=32,
                class_mode='binary')
    return generator

# Copy images to a combined directory structure
def copy_images_to_combined_dir(magnification):
    for train_or_val in ['train', 'val']:
    combined_dir = os.path.join(base_dir, train_or_val, magnification)
    os.makedirs(combined_dir, exist_ok=True)
    for category in ['benign', 'malignant']:
```

```
    category_dir = os.path.join(base_dir, train_or_val, category, magnification)
    for filename in os.listdir(category_dir):
        src_path = os.path.join(category_dir, filename)
        dst_path = os.path.join(combined_dir, category, filename)
        os.makedirs(os.path.join(combined_dir, category), exist_ok=True)
        if not os.path.exists(dst_path):
        os.symlink(src_path, dst_path)

# Define patch and window sizes
patch_sizes = [(32, 32), (64, 64)]
window_sizes = [(64, 64), (128, 128)]

# Loop through each combination of patch and window sizes
for patch_size in patch_sizes:
    for window_size in window_sizes:
    print(f'Training model with patch size {patch_size} and window size {window_size}...')

    # Loop through each magnification and train a model
    for mag in magnifications:
    print(f'Training model for {mag} magnification...')

    # Prepare combined directories
    copy_images_to_combined_dir(mag)

    # Create generators for training and validation
    train_generator = create_generators(mag, 'train', window_size)
    val_generator = create_generators(mag, 'val', window_size)

    # Create and train the model
    input_shape = (window_size[0], window_size[1], 3)
    model = create_model(input_shape)
    history = model.fit(
        train_generator,
        steps_per_epoch=train_generator.samples // 32,
        epochs=10,
        validation_data=val_generator,
        validation_steps=val_generator.samples // 32
    )

    # Evaluate the model
    val_loss, val_acc = model.evaluate(val_generator)
    print(f'\nValidation accuracy for {mag} magnification with patch size {patch_size} and window size
{window_size}: {val_acc}\n')
```

```
Validation accuracy for 40X magnification with patch size (32, 32) and window size (64, 64): 0.8395990133285522
```

```
Validation accuracy for 100X magnification with patch size (32, 32) and window size (64, 64): 0.8417266011238098
```

```
Validation accuracy for 200X magnification with patch size (32, 32) and window size (64, 64): 0.8709677457809448
```

```
Validation accuracy for 400X magnification with patch size (32, 32) and window size (64, 64): 0.8575342297554016
```

```
Validation accuracy for 40X magnification with patch size (32, 32) and window size (128, 128): 0.7092731595039368
```

```
Validation accuracy for 100X magnification with patch size (32, 32) and window size (128, 128): 0.6954436302185059
```

```
Validation accuracy for 200X magnification with patch size (32, 32) and window size (128, 128): 0.8759305477142334
```

```
Validation accuracy for 400X magnification with patch size (32, 32) and window size (128, 128): 0.8739725947380066
```

```
Validation accuracy for 40X magnification with patch size (64, 64) and window size (64, 64): 0.8421052694320679
```

```
Validation accuracy for 100X magnification with patch size (64, 64) and window size (64, 64): 0.7338129281997681
```

```
Validation accuracy for 200X magnification with patch size (64, 64) and window size (64, 64): 0.8387096524238586
```

```
Validation accuracy for 400X magnification with patch size (64, 64) and window size (64, 64): 0.8849315047264099
```

```
Validation accuracy for 40X magnification with patch size (64, 64) and window size (128, 128): 0.8245614171028137
```

```
Validation accuracy for 100X magnification with patch size (64, 64) and window size (128, 128): 0.8561151027679443
```

```
Validation accuracy for 200X magnification with patch size (64, 64) and window size (128, 128): 0.8734491467475891
```

```
Validation accuracy for 400X magnification with patch size (64, 64) and window size (128, 128): 0.8821917772293091
```

AUC-ROC scores

```
Validation accuracy for 40X magnification: 0.7092731595039368

13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 438ms/step

Classification Report for 40X magnification:

              precision    recall  f1-score   support

      Benign       0.33      0.02      0.04       125
   Malignant       0.69      0.98      0.81       274

    accuracy                           0.68       399
   macro avg       0.51      0.50      0.43       399
weighted avg       0.58      0.68      0.57       399

AUC-ROC for 40X magnification: 0.509868613138686
```

```
Validation accuracy for 100X magnification: 0.8609112501144409

14/14 ━━━━━━━━━━━━━━━━━━━━ 6s 435ms/step

Classification Report for 100X magnification:

              precision    recall  f1-score   support

      Benign       0.32      0.30      0.31       129
   Malignant       0.69      0.71      0.70       288

    accuracy                           0.59       417
   macro avg       0.51      0.51      0.51       417
weighted avg       0.58      0.59      0.58       417

AUC-ROC for 100X magnification: 0.5316268303186908
```

```
Validation accuracy for 200X magnification: 0.8684863448143005

13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 460ms/step

Classification Report for 200X magnification:

              precision    recall  f1-score   support

      Benign       0.33      0.31      0.32       125
   Malignant       0.70      0.71      0.70       278

    accuracy                           0.59       403
   macro avg       0.51      0.51      0.51       403
weighted avg       0.58      0.59      0.59       403

AUC-ROC for 200X magnification: 0.5083453237410072
```

```
Validation accuracy for 400X magnification: 0.8849315047264099

12/12 ━━━━━━━━━━━━━━━━━━━━ 6s 473ms/step

Classification Report for 400X magnification:

              precision    recall  f1-score   support

      Benign       0.26      0.23      0.24       118
   Malignant       0.65      0.69      0.67       247

    accuracy                           0.54       365
   macro avg       0.46      0.46      0.46       365
weighted avg       0.52      0.54      0.53       365

AUC-ROC for 400X magnification: 0.45138269402319364
```

SOURCE CODE

```
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report, roc_auc_score

# Define the magnifications and their respective directories
```

```
magnifications = ['40X', '100X', '200X', '400X']
base_dir = '/home/yanajain/Downloads/BreaKHis_Processed'
train_dirs = {mag: [os.path.join(base_dir, 'train', 'benign', mag),
                    os.path.join(base_dir, 'train', 'malignant', mag)] for mag in magnifications}
val_dirs = {mag: [os.path.join(base_dir, 'val', 'benign', mag),
                  os.path.join(base_dir, 'val', 'malignant', mag)] for mag in magnifications}

# Function to create and compile the model
def create_model():
    model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(150, 150, 3)),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam',
        loss='binary_crossentropy',
        metrics=['accuracy'])
    return model

# Helper function to create generators
def create_generators(magnification, train_or_val):
    benign_dir = os.path.join(base_dir, train_or_val, 'benign', magnification)
    malignant_dir = os.path.join(base_dir, train_or_val, 'malignant', magnification)

    datagen = ImageDataGenerator(rescale=1./255,
                rotation_range=20,
                width_shift_range=0.2,
                height_shift_range=0.2,
                shear_range=0.2,
                zoom_range=0.2,
                horizontal_flip=True if train_or_val == 'train' else False)

    combined_dir = os.path.join(base_dir, train_or_val, magnification)
    os.makedirs(combined_dir, exist_ok=True)

    generator = datagen.flow_from_directory(combined_dir,
                    target_size=(150, 150),
                    batch_size=32,
```

```
                                        class_mode='binary')
        return generator

# Copy images to a combined directory structure
def copy_images_to_combined_dir(magnification):
        for train_or_val in ['train', 'val']:
        combined_dir = os.path.join(base_dir, train_or_val, magnification)
        os.makedirs(combined_dir, exist_ok=True)
        for category in ['benign', 'malignant']:
        category_dir = os.path.join(base_dir, train_or_val, category, magnification)
        for filename in os.listdir(category_dir):
                src_path = os.path.join(category_dir, filename)
                dst_path = os.path.join(combined_dir, category, filename)
                os.makedirs(os.path.join(combined_dir, category), exist_ok=True)
                if not os.path.exists(dst_path):
                os.symlink(src_path, dst_path)

# Loop through each magnification and train a model
for mag in magnifications:
        print(f'Training model for {mag} magnification...')

        # Prepare combined directories
        copy_images_to_combined_dir(mag)

        # Create generators for training and validation
        train_generator = create_generators(mag, 'train')
        val_generator = create_generators(mag, 'val')

        # Create and train the model
        model = create_model()
        history = model.fit(
        train_generator,
        steps_per_epoch=train_generator.samples // 32,
        epochs=10,
        validation_data=val_generator,
        validation_steps=val_generator.samples // 32
        )

        # Evaluate the model
        val_loss, val_acc = model.evaluate(val_generator)
        print(f'\nValidation accuracy for {mag} magnification: {val_acc}\n')

        # Obtain true labels and predictions for validation data
        y_true = val_generator.classes
        y_pred_probs = model.predict(val_generator)
```

```
y_pred = (y_pred_probs > 0.5).astype('int32')

# Calculate additional metrics
print(f"\nClassification Report for {mag} magnification:\n")
print(classification_report(y_true, y_pred, target_names=['Benign', 'Malignant']))

# Calculate AUC-ROC
auc_roc = roc_auc_score(y_true, y_pred_probs)
print(f"AUC-ROC for {mag} magnification: {auc_roc}\n")
```

Comparing with CNN model

Main Differences Between a Swin Transformer and a CNN Implementation

1. **Feature Extraction Mechanism:**
    - **Swin Transformer:** Employs self-attention within localized windows to capture both short- and long-range dependencies, enabling it to model complex global relationships within the image.
    - **CNN:** Utilizes convolutional filters to extract local features, gradually building higher-level representations through stacked layers.

2. **Hierarchical Representation:**
    - **Swin Transformer:** Constructs a multi-scale feature hierarchy through patch merging and the shifted window approach, allowing for dynamic and adaptive feature learning.
    - **CNN:** Builds hierarchical features primarily by stacking convolutional layers and pooling operations, which gradually abstract local features into global patterns.

3. **Global Context Modeling:**

    - **Swin Transformer:** Achieves effective global context modeling by shifting attention windows between layers, thereby integrating information from distant regions without excessive computational cost.
    - **CNN:** Focuses on local regions and may require deeper architectures or additional modules (such as global pooling) to capture broader contextual information.

4. **Computational Efficiency:**

    - **Swin Transformer:** Limits self-attention computations to fixed-size windows, significantly reducing the quadratic complexity seen in standard transformers and making it more efficient for high-resolution images.
    - **CNN:** Relies on fixed convolutional operations, which are computationally efficient for detecting local patterns, though they may require additional depth or techniques to achieve similar global context awareness.

Conclusion

The Swin Transformer model using Tensorflow provided better accuracy than the CNN model. However, due to high computational cost and large datasets the client-based server didn't give a drastic result when compared to our previous basic CNN model.