



Machine Learning

LABORATORY: Transformer Homework

NAME:

STUDENT ID#:

Objectives:

- The goal of this assignment is to deepen your understanding of Transformer-based architectures by implementing a Vision Transformer (ViT) model and applying it to a defect detection dataset.
- You will use the ViT model to classify images of industrial defects, treating each image as a sequence of patches.
- Through this assignment, you will:
 - o Understand how the Vision Transformer encodes image patches and uses self-attention to capture global relationships.
 - o Implement the Vision Transformer architecture using PyTorch.
 - o Train the ViT model on a real-world defect dataset.
 - o Evaluate the model's classification performance on unseen test data.

Part 1. Instruction

In this assignment, you will implement a Vision Transformer (ViT) model for image classification using PyTorch, without using any prebuilt ViT modules or high-level Transformer libraries (e.g., no `torchvision.models.vit`, no `timm.create_model`, etc.).

You are required to:

- Manually implement a patch embedding step, dividing each image into non-overlapping patches and flattening them into input sequences.
- Build the core Transformer encoder, including multi-head self-attention, feedforward layers, layer normalization, and residual connections, as described in the original ViT paper.
- Apply a classification head to predict defect categories from the final Transformer outputs.
- Train the model on the provided defect image dataset using a manual training loop.

You should refer to:

- The lecture slide Transformer (page 34 and earlier) for an overview of Vision Transformer design.
- The paper: “An Image is Worth 16×16 Words: Transformers for Image Recognition at Scale” [Dosovitskiy et al., 2020, <https://arxiv.org/pdf/2010.11929>] to understand how ViT works, how to embed image patches, how positional encodings are applied, and how to design the Transformer layers.

You will be provided with the **raw defect dataset**. You are responsible for splitting the dataset yourself into 70% training and 30% testing before starting model training.

General ViT Workflow

- Patch embedding: Split the input image into fixed-size patches, flatten each patch, and apply a linear projection to get patch embeddings.
- Positional encoding: Add learnable positional embeddings to retain spatial order.
- Transformer encoder: Apply multi-head self-attention and feedforward layers across the sequence of patches.



- Classification head: Use the [CLS] token output (or mean pooling) followed by an MLP head to predict the final class label.

After implementing the full forward pass, you must train the model on the defect training set and Evaluate its classification accuracy on the unseen test set.

Hyperparameter Tuning

You are required to adjust the hyperparameters (e.g., learning rate, batch size, number of epochs, etc) to achieve the best possible classification accuracy and help your model reach high performance.

Visualization and Reporting

In your report, you must include the following:

- A screenshot of the model summary and the total number of parameters.
- A screenshot of the final training result, showing total epochs completed, final test loss, and accuracy.
- A confusion matrix visualizing the classification performance.
- 24 example predictions from the test set, with 4 images shown per class.

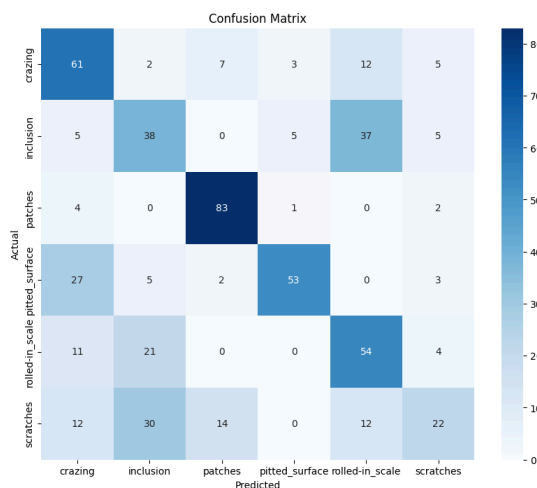
Layer (type:depth-idx)	Param #
ViT	3,264
Sequential: 1-1	--
Rearrange: 2-1	--
Linear: 2-2	1,088
Dropout: 1-2	--
Transformer: 1-3	--
ModuleList: 2-3	--
Identity: 1-4	--
Sequential: 1-5	--
LayerNorm: 2-4	128
Linear: 2-5	390
Total params: 499,462	
Trainable params: 499,462	
Non-trainable params: 0	

```
Epoch: 5
[ 0/60000 ( 0%)] Loss: 0.0816
[10000/60000 ( 17%)] Loss: 0.1146
[20000/60000 ( 33%)] Loss: 0.0733
[30000/60000 ( 50%)] Loss: 0.0584
[40000/60000 ( 67%)] Loss: 0.1069
[50000/60000 ( 83%)] Loss: 0.0777
```

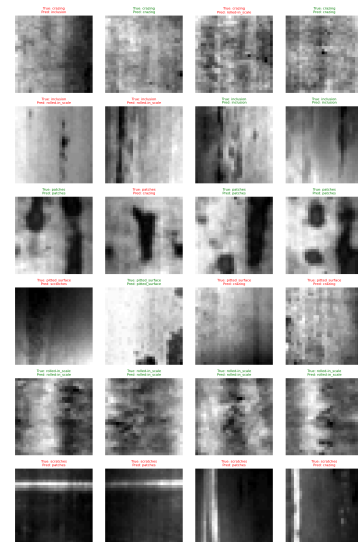
Average test loss: 0.1183 Accuracy: 9603/10000 (96.03%)

(a)

(b)



(c)



(d)



Part 2. Code Template

Step	Procedure
1	<pre> # ===== # Step 1: Unzip Dataset # ===== import zipfile import os # TODO: Set the correct uploaded ZIP filename zip_path = 'dataset.zip' # <-- replace with the exact uploaded filename extract_path = './dataset' # folder where you want to extract # Unzip the dataset with zipfile.ZipFile(zip_path, 'r') as zip_ref: zip_ref.extractall(extract_path) print(f"✅ Unzipped to: {os.path.abspath(extract_path)}") # Optional: List the extracted folder contents print("Contents:") print(os.listdir(extract_path)) # ===== # Step 2: Split Dataset into Train/Test # ===== import shutil import random # TODO: Set source folder (where unzipped dataset is) and target folder (where split dataset will go) SOURCE_DIR = 'dataset' # folder from unzip TARGET_DIR = 'dataset_split' # new folder to store split data # Split ratios train_ratio = 0.7 test_ratio = 0.3 # note: you can calculate this as 1 - train_ratio if needed # Set random seed for reproducibility random.seed(42) # Create target train/test directories per class for split in ['train', 'test']: for class_name in os.listdir(SOURCE_DIR): os.makedirs(os.path.join(TARGET_DIR, split, class_name), exist_ok=True) # Process each class folder for class_name in os.listdir(SOURCE_DIR): class_path = os.path.join(SOURCE_DIR, class_name) if not os.path.isdir(class_path): </pre>



	<pre> continue images = os.listdir(class_path) random.shuffle(images) # Calculate split point train_cutoff = int(len(images) * train_ratio) # Split images train_images = images[:train_cutoff] test_images = images[train_cutoff:] # Copy training images for img_name in train_images: src = os.path.join(class_path, img_name) dst = os.path.join(TARGET_DIR, 'train', class_name, img_name) shutil.copyfile(src, dst) # Copy testing images for img_name in test_images: src = os.path.join(class_path, img_name) dst = os.path.join(TARGET_DIR, 'test', class_name, img_name) shutil.copyfile(src, dst) print("✅ Dataset split complete!") </pre>
2	<pre> import os import torch from torch.utils.data import DataLoader import torchvision import torchvision.transforms as transforms import matplotlib.pyplot as plt # Set dataset paths (update if needed) TRAIN_PATH = 'dataset_split/train' TEST_PATH = 'dataset_split/test' # Define image transforms: resize, grayscale, tensor, normalize transform_custom = transforms.Compose([transforms.Resize((28, 28)), transforms.Grayscale(num_output_channels=1), transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))]) # Load datasets train_set = torchvision.datasets.ImageFolder(root=TRAIN_PATH, transform=transform_custom) test_set = torchvision.datasets.ImageFolder(root=TEST_PATH, transform=transform_custom) # Print dataset info print("Classes:", train_set.classes) print("Train samples:", len(train_set)) print("Test samples:", len(test_set)) # Show example images (2 per class) </pre>



	<pre> def show_2x6_grid(dataset, n_per_class=2, title="Example Grid"): class_counts = {i: 0 for i in range(len(dataset.classes))} collected = {i: [] for i in range(len(dataset.classes))} for img, label in dataset: if class_counts[label] < n_per_class: collected[label].append(img) class_counts[label] += 1 if all(c >= n_per_class for c in class_counts.values()): break fig, axes = plt.subplots(n_per_class, len(dataset.classes), figsize=(len(dataset.classes)*2, n_per_class*2)) for col, imgs in collected.items(): for row in range(n_per_class): ax = axes[row][col] if n_per_class > 1 else axes[col] img = imgs[row].numpy().transpose(1, 2, 0) * 0.5 + 0.5 # unnormalize ax.imshow(img.squeeze(), cmap='gray') ax.set_title(dataset.classes[col], fontsize=8) ax.axis('off') plt.suptitle(title) plt.tight_layout() plt.show() # Show training and test grids show_2x6_grid(train_set, 2, "Train Set Grid") show_2x6_grid(test_set, 2, "Test Set Grid") </pre>
3	<pre> import torch import torch.nn as nn from einops import rearrange, repeat, einsum # === Helper === def pair(t): return t if isinstance(t, tuple) else (t, t) # === TODO 1: Define PreNorm block === class PreNorm(nn.Module): def __init__(self, dim, fn): super().__init__() # TODO: initialize LayerNorm and store fn def forward(self, x, **kwargs): # TODO: apply LayerNorm + fn pass # === TODO 2: Define FeedForward block === class FeedForward(nn.Module): def __init__(self, dim, hidden_dim, dropout=0.): super().__init__() # TODO: define two Linear layers + activation + dropout def forward(self, x): # TODO: apply layers pass </pre>



```

# === TODO 3: Define Attention block ===
class Attention(nn.Module):
    def __init__(self, dim, heads=4, dim_head=64, dropout=0.):
        super().__init__()
        # TODO: set up qkv projections, softmax attention, final output projection

    def forward(self, x):
        # TODO: compute q, k, v, attention, and output
        pass

# === TODO 4: Define Transformer Encoder ===
class Transformer(nn.Module):
    def __init__(self, dim, depth, heads, dim_head, mlp_dim, dropout=0.):
        super().__init__()
        # TODO: stack multiple PreNorm + Attention + FeedForward layers

    def forward(self, x):
        # TODO: pass through each Transformer layer
        pass

# === TODO 5: Define Vision Transformer (ViT) ===
class ViT(nn.Module):
    def __init__(self, *, image_size, patch_size, num_classes, dim, depth, heads,
                 mlp_dim, pool='cls', channels=1, dim_head=64, dropout=0., emb_dropout=0.):
        super().__init__()
        # TODO: calculate patch numbers, set up patch embedding, positional embedding, cls token,
        # transformer, mlp head

    def forward(self, img):
        # TODO: apply patch embedding, add cls token + pos embedding, run transformer, pool, mlp head
        pass

# === TODO 6: Initialize model ===
# Example hyperparameters (students should adjust!)
model = ViT(
    image_size=28,
    patch_size=4,
    num_classes=6,
    channels=1,
    dim=64,
    depth=6,
    heads=4,
    mlp_dim=128
)

# === TODO 7: Set up optimizer ===
import torch.optim as optim
optimizer = optim.Adam(model.parameters(), lr=0.003)

# === TODO 8: Print or summarize the model ===
print(model)
# Optionally: from torchsummary import summary

```



	# summary(model, input_size=(1, 28, 28))
4	<pre> # === TODO 1: Define training epoch === def train_epoch(model, optimizer, data_loader, loss_history): model.train() total_samples = len(data_loader.dataset) for i, (data, target) in enumerate(data_loader): # TODO: Zero gradients # TODO: Forward pass # TODO: Compute loss # TODO: Backward pass and optimizer step if i % 100 == 0: # TODO: Print progress info and save loss pass # === TODO 2: Define evaluation function === def evaluate(model, data_loader, loss_history): model.eval() total_samples = len(data_loader.dataset) correct_samples = 0 total_loss = 0 with torch.no_grad(): for data, target in data_loader: # TODO: Forward pass # TODO: Compute loss # TODO: Get predictions and count correct samples # TODO: Compute average loss and accuracy # TODO: Print evaluation summary </pre>
5	<pre> # === SET EPOCHS === N_EPOCHS = 1 # === START TIMER === start_time = time.time() # === INIT LOSS TRACKERS === train_loss_history, test_loss_history = [], [] # === MAIN TRAINING LOOP === for epoch in range(1, N_EPOCHS + 1): print('Epoch:', epoch) train_epoch(model, optimizer, train_loader, train_loss_history) evaluate(model, test_loader, test_loss_history) # === PRINT TOTAL TIME === print('Execution time:', '{:5.2f}'.format(time.time() - start_time), 'seconds') # === SAVE TRAINED MODEL === torch.save(model.state_dict(), 'Student_ID.pth') # replace with your actual Student ID print("✅ Model saved as .pth") </pre>
	#LOAD MODEL - if needed



	<pre> # Make sure you define the same ViT model structure first model = ViT(image_size=28, patch_size=4, num_classes=6, channels=1, dim=64, depth=6, heads=4, mlp_dim=128) # Load saved weights model.load_state_dict(torch.load('Student_ID.pth')) model.eval() print("✅ Model loaded and ready for testing") </pre>
6	<pre> import torch # === TODO: Define function to plot confusion matrix === def plot_confusion_matrix(model, data_loader, class_names): # HINT: # - Get predictions and true labels # - Compute confusion matrix (sklearn) # - Plot with seaborn heatmap pass # === TODO: Define function to plot example predictions === def plot_classwise_predictions(model, data_loader, class_names, samples_per_class=4): # HINT: # - Collect a few correct/incorrect predictions per class # - Plot grid of images with true vs predicted labels pass # === TODO: After training, call both functions === # plot_confusion_matrix(model, test_loader, train_set.classes) # plot_classwise_predictions(model, test_loader, train_set.classes, samples_per_class=4) </pre>

Grading Assignment & Submission

Implementation:

1. **(35%) Vision Transformer Model Design:** Implement the full ViT model from scratch, including all required components, without using any prebuilt ViT libraries; code must compile and run without critical errors.
2. **(15%) Training and Hyperparameter Tuning:** Train the model on the defect dataset and tune hyperparameters to achieve at least 60% test accuracy, with higher accuracy earning more points.
3. **(5%) Evaluation Function:** Write a correct evaluation loop to compute and report the test set's accuracy and loss.
4. **(10%) Visualization:** Provide a clear confusion matrix plot and 24 example predictions (4 per class) showing both true and predicted labels.
5. **(5%) Report Quality:** Submit a well-organized report summarizing your implementation, results, and analysis.

Question:

6. **(5%)** Briefly explain the role of patch embedding and positional encoding in ViT. You may include parts of your Step 3 model code to support your explanation.



7. **(5%)** Describe which hyperparameters you tuned, why you chose them, and how they affected your final accuracy.
8. **(5%)** Compare ViT and CNN for image classification: what are the main differences, and when might one be preferred over the other using the provided dataset?
9. **(5%)** Report the final achieved test accuracy; explain whether you reached the >60% baseline — if not, describe what you tried and why it might have failed; if you did, explain how you achieved it.
10. (10%) Based on the paper you referred to (Dosovitskiy et al., An Image is Worth 16x16 Words), please brief explain: *You may include parts of your Step 3 model code to support your explanation.*
 - a. How does the Vision Transformer process input images from start to finish?
 - b. How are the image patches divided and transformed into input sequences?
 - c. How does the multi-head self-attention mechanism operate within the Transformer encoder?
 - d. How does the model use the [CLS] token (or final output) to produce the final image classification?

Submission :

1. Upload both your report, code, and trained model to the E3 system (Lab8 Homework). Name your files correctly as follows:
 - a. Report: StudentID_Lab8_Homework.pdf
 - b. Code: StudentID_Lab8_Homework.py or StudentID_Lab8_Homework.ipynb
 - c. Trained Model: StudentID_Lab8_Homework.pth
2. ⚠ Important: Make sure all three files are uploaded. Missing even one will result in no grade for this assignment.
3. Deadline: May 13 - 21:00 PM
4. Plagiarism is **strictly prohibited**. Submitting copied work from other students will result in penalties.
5. References: You must include any references or external sources you used when working on this assignment in your report.

Results and Discussion:

