BME646 and ECE60146: Homework 9

Spring 2026
Due Date: Tuesday, April 14, 2026, 11:59 pm ET
TA: Somosmita Mitra (mitra26@purdue.edu)

Turn in typed solutions via Gradescope. Post questions to Piazza. Additional instructions can be found at the end. **Late submissions will be accepted with penalty: -10 points per late day, up to 5 days.**

1 Introduction

In HW8 you worked with generative models: DCGANs, diffusion, and Stable Diffusion. All of those architectures used convolutional layers (or UNets built from them) as their primary computational building blocks.

In this homework you will work with a different architectural element: **attention**. You will experiment with the Transformer, a network that replaces convolutions and recurrence entirely with attention-based computations. The paper that introduced this idea, "Attention is All You Need" by Vaswani et al. [3], appeared in 2017 and has since become the basis for nearly all large-scale language and vision models.

The homework has four tasks.

In Task 1 you will run DLStudio's `TransformerFG` on English-to-Spanish translation, study the encoder-decoder architecture, and see how the self-attention and cross-attention mechanisms interact during sequence-to-sequence learning.

In Task 2 you will run DLStudio's `visTransformer` on CIFAR-10 as a warm-up, study how images are decomposed into patch sequences, and understand the class token and positional encoding mechanisms.

In Task 3 you will adapt DLStudio's `visTransformer` to a 20-class subset of the Food-101 dataset, train it from scratch, then modify the `AttentionHead` code to extract and visualize attention weight heatmaps on food images. You will compare attention patterns between visually similar food categories (e.g., different pasta dishes, different types of steak) to see whether the model attends to different regions when distinguishing them.

In Task 4 you will download a pretrained Vision Transformer, fine-tune it on the same 20-class Food-101 subset, and compare its accuracy against your from-scratch model from Task 3. This comparison illustrates why pretraining on large datasets matters, a topic the next lecture on BERT and GPT will cover in depth.

1.1 Quick reference: lecture slide numbers

Use this table to look up the lecture material for each topic.

HW9 topic	**Lecture**	**Slides**
Transformers for seq2seq		
Dot-product attention and QKV	Transformers [2]	11–25
Multi-headed attention	Transformers	26–29
AttentionHead and SelfAttention code	Transformers	30–35
Encoder-decoder architecture overview	Transformers	36–41
MasterEncoder, BasicEncoder	Transformers	42–46
CrossAttention, CrossAttentionHead	Transformers	47–52
BasicDecoder, MasterDecoder with masking	Transformers	53–59
Positional encoding	Transformers	60–64
TransformerFG vs. TransformerPreLN	Transformers	65–69
Training difficulty, BLEU metric	Transformers	70–75
English-Spanish results	Transformers	76–86
Vision Transformers		
ViT architecture and patch embeddings	Transformers	91–98
visTransformer code in DLStudio	Transformers	99–104
QKV for inter-pixel attention	Transformers	105–108

Which slides to read for each task:

Task	**Key slides**
Task 1: Seq2seq warm-up	Transformers 11–86
Task 2: Vision Transformer warm-up	Transformers 91–104
Task 3: Food-101 from scratch + attention viz	Transformers 30–35, 91–104
Task 4: Pretrained ViT fine-tuning	Transformers 91–104

2 Setting up your environment

2.1 Prerequisites from previous homeworks

Ensure you have:

- DLStudio version 2.5.6 installed, including the `Transformers` module
- Python 3.8+ with PyTorch, NumPy, Matplotlib installed
- The English-Spanish dataset for seq2seq (see Section 2.2)
- CIFAR-10 (downloads automatically via `torchvision`)

2.2 Downloading the English-Spanish dataset

DLStudio includes the data archive `en_es_xformer_8_90000.tar.gz`. The number 8 refers to the maximum number of words per sentence (10 including SOS and EOS tokens). The number 90,000 is the number of English-Spanish sentence pairs.

Download the archive from the DLStudio page [1]. Place it in the `ExamplesTransformers` directory and extract it:

```
tar zxvf en_es_xformer_8_90000.tar.gz
```

2.3 The Food-101 subset (Tasks 3–4)

Tasks 3 and 4 both use a **20-class subset** of the Food-101 dataset [5]. The full dataset contains 101,000 images across 101 food categories (750 training and 250 test images per class). You will select 20 classes, giving you 15,000 training images and 5,000 test images.

The full dataset downloads automatically via `torchvision`. Use the following code to select a 20-class subset. You **must use these exact 20 classes** so that grading is consistent:

```
from torchvision.datasets import Food101
import torchvision.transforms as T
from torch.utils.data import Subset

FOOD20_CLASSES = [
    'caesar_salad',      # 0
    'cheesecake',        # 1
    'chicken_wings',     # 2
    'chocolate_cake',    # 3
    'chocolate_mousse',  # 4
    'club_sandwich',     # 5
    'donuts',            # 6
    'filet_mignon',      # 7
    'fish_and_chips',    # 8
    'french_fries',      # 9
    'fried_rice',        # 10
    'grilled_salmon',    # 11
    'hamburger',         # 12
    'ice_cream',         # 13
    'omelette',          # 14
    'pizza',             # 15
    'ramen',             # 16
    'spaghetti_bolognese',   # 17
    'spaghetti_carbonara',   # 18
    'steak',             # 19
```

```
]

def make_food20(split, transform):
    full = Food101(root='./data',
        split=split, download=True,
        transform=transform)
    # Food101 class_to_idx maps class name
    # to original label (0-100)
    orig_labels = [
        full.class_to_idx[c]
        for c in FOOD20_CLASSES]
    indices = [
        i for i, (_, lbl)
        in enumerate(full._labels)
        if lbl in orig_labels]
    # Remap labels to 0-19
    label_map = {
        orig: new for new, orig
        in enumerate(orig_labels)}
    subset = Subset(full, indices)
    return subset, label_map
```

Note: The full Food-101 download is approximately 5 GB, even though you will only use 20 of the 101 classes. Start this download early. If using Google Colab, download to your Google Drive to avoid re-downloading each session.

The 20 classes above were chosen to include several visually similar pairs for the attention similarity analysis in Task 3: "chocolate cake" vs. "chocolate mousse", "spaghetti bolognese" vs. "spaghetti carbonara", "filet mignon" vs. "steak", and "caesar salad" vs. "omelette" (both plated with similar colors).

2.4 Setting up the pretrained ViT (Task 4)

Task 4 requires downloading a pretrained Vision Transformer. Install the required package:

```
pip install timm
```

The `timm` (PyTorch Image Models) library provides pretrained ViT weights. The first time you load a model, it downloads the weights (∼330 MB for ViT-Base):

```
import timm

model = timm.create_model(
    'vit_base_patch16_224',
```

```
    pretrained=True,
    num_classes=20)
```

GPU recommended for Tasks 3 and 4. Use Google Colab if you do not have a local GPU.

3 Programming tasks (100 points total)

3.1 Task 1: Seq2seq translation warm-up (15 points)

Objective: Run DLStudio's TransformerFG on the English-Spanish dataset, study the architecture, and understand the self-attention and cross-attention mechanisms.

3.1.1 Running TransformerFG (7 points)

Run `seq2seq_with_transformerFG.py` in the `ExamplesTransformers` directory. Train for **4 epochs** using the parameters shown on Slide 78 of the lecture [2] (batch size 50, embedding size 256, 4 Basic Encoders, 4 Basic Decoders, 4 Attention Heads, 4000 warm-up steps).

After training, test the model on 20 randomly selected sentence pairs from the dataset. Record which translations are correct and which contain errors.

With only 4 epochs, the model will still be learning and many translations will be partially or fully incorrect. That is expected. The goal here is to observe the training dynamics and the warm-up phase, not to produce perfect translations.

Deliverables:

- Training loss curve over the 4 epochs
- The 20 translations with labels (correct, semantically correct, wrong)
- Translation accuracy as a fraction (e.g., 5/20 correct)
- A paragraph describing how the loss behaves during the warm-up phase vs. after warm-up

3.1.2 Understanding the architecture and training (8 points)

Study the TransformerFG code and the lecture slides. **Report** (answer each in 1–2 paragraphs):

- **Self-attention computation:** Referring to Eq. (3) on Slide 20, explain how the $Q \cdot K^T$ dot-product produces an $N_w \times N_w$ attention map. What does row i of this matrix represent after the `nn.Softmax` normalization? Why is the result divided by $\sqrt{M}$?

- **Multi-headed attention:** On Slide 29, the embedding axis is split into `num_atten_heads` slices. If the embedding size is 256 and you use 4 attention heads, what is the size of each slice (`sqkv`)? Why might multiple heads be better than a single head that operates on the full embedding?

- **TransformerFG vs. TransformerPreLN:** Looking at the diagrams on Slide 68, describe where LayerNorm is applied in each variant. Why does TransformerFG require a learning-rate warm-up while TransformerPreLN does not? (See Slides 70–75.)

Grading:

- Successful training with loss curve and translations (7 points)

- Clear answers to the three questions (8 points)

3.2 Task 2: Vision Transformer warm-up on CIFAR-10 (15 points)

Objective: Run DLStudio's `visTransformer` on CIFAR-10 to verify that the code works and to understand how images become patch sequences.

3.2.1 Running the visTransformer (7 points)

Run `image_recog_with_visTransformer.py` in the `ExamplesTransformers` directory. Train for at least 40 epochs on CIFAR-10.

After training, run `test_checkpoint_for_visTransformer.py` to evaluate the best checkpoint on the test set.

Deliverables:

- Training loss curve over 40 epochs

- The 10×10 confusion matrix on the test set (same format as Slide 103)

- Overall test accuracy

3.2.2 Understanding the ViT architecture (8 points)

Study the visTransformer code (Slides 99–102). **Report** (answer each in 1–2 paragraphs):

- **Patch embedding:** CIFAR-10 images are $32 \times 32 \times 3$. If the patch size is 16×16, how many patches P does each image produce? What is the dimensionality of the raw pixel data in each patch before embedding? After the `nn.Linear` embedding layer on Slide 102 maps each patch to embedding size M, what is the shape of the resulting tensor (ignoring the batch axis)?

- **The class token:** On Slide 100, Line (3) creates a learnable class token, and Line (5) concatenates it with the patch embeddings. After this concatenation, what is the shape of the tensor fed into the MasterEncoder? At the output, Line (7) keeps only the class token embedding. Explain why the class token, after passing through all the self-attention layers, contains enough information to predict the image class.

- **Positional encoding for patches:** On Slide 102, Line (2) creates learnable positional encodings and Line (5) adds them to the patch embeddings. Why are positional encodings necessary for a ViT? What would happen if you removed them? (Hint: the self-attention mechanism treats the input as a set, not a sequence, unless position information is injected.)

Grading:

- Successful training with loss curve, confusion matrix, and accuracy (7 points)

- Clear answers to the three questions (8 points)

3.3 Task 3: Train visTransformer on Food-101 subset and visualize attention (25 points)

Objective: Adapt DLStudio's visTransformer to the 20-class Food-101 subset, train it from scratch, then extract and visualize the attention weight matrices to see which regions of food images the network attends to when classifying. Compare attention patterns between visually similar food categories.

3.3.1 Adapting and training visTransformer on Food-101 (8 points)

Copy the visTransformer code from DLStudio into your own file. Modify it to work with the 20-class Food-101 subset (see Section 2.3) with images resized to 64×64. The changes you need to make:

- Set `num_classes` to 20
- Use a patch size of 8×8, which gives $P = 64$ patches per image and an attention map of size 65×65 (64 patches + 1 class token)
- Update the `image_size` and `patch_size` parameters accordingly
- Update the final `nn.Linear` layer to output 20 classes
- Use the data loading code from Section 2.3, remapping labels to 0–19

Train for at least 20 epochs. With 20 classes and a small model, accuracy will be modest (20–50% is realistic). Random guessing over 20 classes would give 5%, so anything well above that shows the model is learning.

Deliverables:

- Your adapted visTransformer code
- Training loss curve over epochs
- Test accuracy
- A paragraph describing the result: did the model converge? How does the accuracy compare to random guessing?

3.3.2 Extracting attention weights (7 points)

Modify the `AttentionHead` class so that it stores the attention weights during a forward pass:

1. In `AttentionHead.forward()`, after computing the softmax-normalized $Q \cdot K^T$ matrix (Line (L) on Slide 32), store the result as an instance attribute:

```
self.attention_weights = \
    rowwise_softmax_normalizations.detach()
```

2. In `SelfAttention.forward()`, after calling each attention head, collect its stored weights into a list.

3. Thread these weights up through BasicEncoder and MasterEncoder so the top-level model can return them.

You only need to collect attention weights at inference time, not during training. Use a `torch.no_grad()` block.

Deliverables:

- Your modified AttentionHead, SelfAttention, BasicEncoder, and MasterEncoder code
- A brief description (3–5 sentences) of what you changed

3.3.3 Attention visualization and similarity analysis (10 points)

Using your trained Food-101 visTransformer checkpoint, extract and visualize the attention maps.

Part A – Individual attention maps. Pick 4 correctly classified test images from 4 different food categories. For each image:

1. Plot the full 65×65 attention heatmap from the **last BasicEncoder** for one attention head. Use `matplotlib.pyplot.imshow` with a sequential colormap.
2. From the class token's row (row 0) of the attention matrix, extract the 64 weights corresponding to the image patches. Reshape them into the 8×8 spatial patch grid. Upsample this grid to 64×64 using nearest-neighbor or bilinear interpolation and overlay it on the original image as a semi-transparent heatmap. This shows which regions the class token attends to when predicting the food category.

Part B – Similarity analysis. Choose 2 pairs of visually similar food categories from the 20-class subset. Good pairs include: "spaghetti bolognese" vs. "spaghetti carbonara", "filet mignon" vs. "steak", "chocolate cake" vs. "chocolate mousse". For each pair:

1. Show the class-token attention overlay for one image from each category, side by side.
2. Write 3–5 sentences: Does the model attend to different regions when distinguishing these two similar categories? For example, does it focus on the sauce for pasta dishes, or on the cut of meat for steak variants? If the model misclassifies one of the pair, does the attention map give any clue as to why?

Deliverables:

- Part A: For each of the 4 images, show the original, the full heatmap, and the spatial overlay. Write 2–3 sentences per image describing where the class token attends.
- Part B: For each of the 2 pairs, show side-by-side overlays with 3–5 sentences of comparative analysis.

Grading:

- Adapted visTransformer with training on Food-101 subset (8 points)
- Correct attention extraction code (7 points)
- Visualizations and similarity analysis (10 points)

3.4 Task 4: Pretrained ViT fine-tuning on Food-101 subset (45 points)

Objective: Download a pretrained ViT-Base model, fine-tune it on the same 20-class Food-101 subset, and compare its accuracy against your from-scratch visTransformer from Task 3.

3.4.1 Background: why pretraining matters

On Slide 95 of the lecture [2], Dosovitskiy et al. observed that if you chop an image into 16×16 patches and represent each patch with a learnable embedding, you can apply the same transformer architecture used for language to image recognition.

The ViT paper also showed that training a Vision Transformer from scratch on a small or medium dataset produces poor results. The model needs to learn both the patch representations and the inter-patch relationships at the same time, which requires a lot of data. When pretrained on a large dataset (ImageNet-21k, with 14 million images and 21,000 classes), the same ViT architecture achieves state-of-the-art accuracy after fine-tuning on smaller target datasets.

This is the same principle behind BERT and GPT, which you will study in next week's lecture: inexpensive unsupervised pretraining on large corpora, followed by cheap supervised fine-tuning on smaller task-specific datasets.

3.4.2 Fine-tuning ViT-Base on the Food-101 subset (25 points)

Download the pretrained `vit_base_patch16_224` model from the `timm` library (see Section 2.4). This model was pretrained on ImageNet-1k (1.28 million images, 1000 classes).

Fine-tune it on your 20-class Food-101 subset with the following configuration:

- Replace the classification head: `num_classes=20`
- Resize all images to 224×224 (the resolution this ViT was pretrained at)
- Use standard ImageNet normalization (`mean=[0.485, 0.456, 0.406]`, `std=[0.229, 0.224, 0.225]`)
- Apply data augmentation during training: random resized crop to 224, random horizontal flip
- At test time: resize to 256, center crop to 224

Training configuration (suggested):

- Optimizer: `AdamW(lr=5e-5, weight_decay=0.01)`
- Batch size: 32
- Epochs: 5–10
- Loss: `nn.CrossEntropyLoss`
- Optionally freeze the patch embedding and early transformer blocks for the first few epochs, then unfreeze for full fine-tuning

```
import timm
import torch
import torch.nn as nn
from torchvision.datasets import Food101
import torchvision.transforms as T
from torch.utils.data import Subset

# Load pretrained ViT
model = timm.create_model(
    'vit_base_patch16_224',
    pretrained=True,
    num_classes=20)
```

```
model = model.to(device)

# Data transforms
train_tf = T.Compose([
    T.RandomResizedCrop(224),
    T.RandomHorizontalFlip(),
    T.ToTensor(),
    T.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]),
])
test_tf = T.Compose([
    T.Resize(256),
    T.CenterCrop(224),
    T.ToTensor(),
    T.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]),
])

# Use the same 20-class subset
# and label remapping from Section 2.3
train_subset, label_map = make_food20(
    'train', train_tf)
test_subset, _ = make_food20(
    'test', test_tf)

train_loader = torch.utils.data.DataLoader(
    train_subset, batch_size=32, shuffle=True)
test_loader = torch.utils.data.DataLoader(
    test_subset, batch_size=32, shuffle=False)

# Remember to remap labels using label_map
# in your training and evaluation loops

optimizer = torch.optim.AdamW(
    model.parameters(),
    lr=5e-5, weight_decay=0.01)
criterion = nn.CrossEntropyLoss()

for epoch in range(5):
    model.train()
    for images, labels in train_loader:
        labels = torch.tensor(
            [label_map[l.item()]
             for l in labels])
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
```

```
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Deliverables:

- Your complete fine-tuning script
- Training loss curve and validation accuracy curve over epochs
- Test accuracy on the Food-101 subset test split
- A 4×4 grid showing 16 test images with their predicted and true labels (mark incorrect predictions in red)
- The number of learnable parameters in the model

3.4.3 Comparison and analysis (20 points)

Compare the fine-tuned ViT-Base against the DLStudio visTransformer you trained from scratch in Task 3. Both were trained on the same 20-class Food-101 subset.

Deliverables:

- A comparison table:

Model	**Size**	**Pretr.?**	**Accuracy**	**Params**
visTransformer (Task 3)	64^2	No	?	?
ViT-Base fine-tuned (Task 4)	224^2	Yes	?	?

- **Report** (answer each in 1–2 paragraphs):
  - **Accuracy gap:** How large is the difference in test accuracy between the two models? Both are Transformer architectures trained on the same 20 food categories, but they differ in model size, image resolution, and whether the weights were pretrained. Which of these three factors do you think contributes the most to the accuracy gap? Why?
  - **Model capacity and data requirements:** ViT-Base has roughly 86 million parameters. DLStudio's visTransformer is much smaller. The 20-class subset has 15,000 training images. What would you expect to happen if you trained ViT-Base from scratch (no pre-training) on this subset? Why would the result likely be worse

than what the small DLStudio model achieves? (You do not need to run this experiment; reason about it from what you know about overfitting.)

- **Connection to language models:** The next lecture will cover BERT and GPT, which also use Transformer architectures and large-scale pretraining. Based on your experience in this homework, explain in your own words why pretraining on a large generic dataset and then fine-tuning on a smaller task-specific dataset works well. What parallels do you see between pretraining a ViT on ImageNet and pretraining a language model on a large text corpus?

Grading:

- Fine-tuning script, training curves, test accuracy, and image grid (25 points)
- Comparison table and analysis paragraphs (20 points)

4 Submission instructions

4.1 What to submit in the report

Read this section carefully. Incomplete submissions will lose points. Every item listed below must be present in your submission.

1. **Task 1: Seq2seq warm-up (15 points)**
   (a) **Code:** Script used to run TransformerFG
   (b) **Figure:** Training loss curve over 4 epochs
   (c) **Text:** 20 translations with correctness labels and accuracy fraction
   (d) **Text:** Warm-up phase observations
   (e) **Text:** Self-attention computation explanation
   (f) **Text:** Multi-headed attention explanation
   (g) **Text:** TransformerFG vs. TransformerPreLN comparison
2. **Task 2: Vision Transformer warm-up (15 points)**
   (a) **Code:** Script used to run visTransformer

(b) **Figure:** Training loss curve over 40 epochs

(c) **Table:** 10×10 confusion matrix on test set

(d) **Text:** Overall test accuracy

(e) **Text:** Patch embedding explanation

(f) **Text:** Class token explanation

(g) **Text:** Positional encoding explanation

3. **Task 3: Food-101 from scratch + attention visualization (25 points)**

   (a) **Code:** Adapted visTransformer for Food-101 subset

   (b) **Figure:** Training loss curve

   (c) **Text:** Test accuracy and training observations

   (d) **Code:** Modified AttentionHead, SelfAttention, BasicEncoder, MasterEncoder

   (e) **Text:** Description of attention extraction modifications

   (f) **Figure:** Part A: 4 images with full heatmap and spatial overlay each

   (g) **Text:** Part A: Per-image attention analysis (2–3 sentences each)

   (h) **Figure:** Part B: 2 pairs of similar categories with side-by-side overlays

   (i) **Text:** Part B: Comparative analysis per pair (3–5 sentences each)

4. **Task 4: Pretrained ViT fine-tuning (45 points)**

   (a) **Code:** Fine-tuning script for ViT-Base on Food-101 subset

   (b) **Figure:** Training loss and validation accuracy curves

   (c) **Text:** Test accuracy and parameter count

   (d) **Figure:** 4×4 image grid with predictions

   (e) **Table:** Two-model comparison table

   (f) **Text:** Accuracy gap discussion

   (g) **Text:** Model capacity and data requirements discussion

   (h) **Text:** Connection to language model pretraining

4.2 Code files to submit

1. `attention_viz.py`: Adapted visTransformer for Food-101 subset with attention extraction and visualization code
2. `finetune_vit.py`: ViT-Base fine-tuning on Food-101 subset
3. `requirements.txt`: Dependencies (must include `timm`, `torchvision`)
4. `README.md`: Instructions to run your code, including compute setup and model download instructions

4.3 Autograder interface requirements

The autograder will import your `attention_viz.py` file and call the following function. If the file name or function signature does not match exactly, the autograder will fail and you will lose points.

4.3.1 Required file names

Submit exactly this file name (case-sensitive):

- `attention_viz.py`

4.3.2 `attention_viz.py`

Must contain:

- `extract_attention_weights(model, input_tensor)`: Takes a modified visTransformer model and an input tensor, runs a forward pass, and returns a list of attention weight tensors (one per BasicEncoder layer). Each tensor should have shape `(num_atten_heads, seq_len, seq_len)` where `seq_len` is the number of patches plus the class token.

The autograder will run:

```
from attention_viz import \
    extract_attention_weights
import torch

# Test with dummy input
# 1 batch, 5 positions (4 patches + 1 cls),
# 128 embedding
dummy_input = torch.randn(1, 5, 128)

# Assume model has been constructed with
```

```
# num_atten_heads=4, num_basic_encoders=2
weights = extract_attention_weights(
    model, dummy_input)

assert isinstance(weights, list)
assert len(weights) == 2  # 2 basic encoders
assert weights[0].shape == (4, 5, 5)
# 4 heads, 5x5 attention map
assert weights[1].shape == (4, 5, 5)

# Each row should sum to ~1 (softmax)
import numpy as np
for w in weights:
    row_sums = w.sum(dim=-1).numpy()
    assert np.allclose(row_sums, 1.0,
                       atol=1e-5)
```

4.3.3 Autograder summary

File	Required names	Autograder tests
`attention_viz.py`	`extract_attention_weights`	Shape checks, list length, softmax row-sum validation

Important notes:

- Do **not** rename the file or function.
- Do **not** put executable code at the module level. Guard scripts with `if __name__ == '__main__':` blocks.
- **Cite DLStudio source code** wherever you borrow or adapt code.

4.4 Code quality requirements

- Well-commented code explaining what each section does
- Meaningful variable and function names
- Docstrings for all classes and functions
- Code should run without modification (given correct paths and model downloads)
- Follow PEP 8 style guidelines

4.5 Report formatting

- **Include code snippets in the report for every task.** For each task, show the code you wrote alongside the outputs it produced. If the code snippet is not present, you will lose points on that task.
- Place output directly next to the corresponding code block. Avoid placing code and output in separate sections.
- Use figures and tables with captions
- Number equations, figures, and tables
- Use proper citations when referencing papers or DLStudio

4.6 Additional guidelines

- Convert Jupyter notebooks to .py files. **Do NOT submit .ipynb**
- If submitting late, submit only once
- **Do NOT submit dataset files or model weights**, only code
- Your code and report must be your own work
- Document your compute setup (CPU/GPU, estimated training times) in your README

5 Frequently asked questions (FAQ)

Q: Can I reuse code from previous homeworks?

A: Yes. You are encouraged to reuse plotting utilities, dataset loading code, etc.

Q: TransformerFG training is unstable or the loss is not decreasing.

A: Make sure you are using the learning-rate warm-up as described on Slides 70–75. With only 4 epochs the model may still be in the early learning stages. If the loss stays completely flat, try a different random seed.

Q: Only 4 epochs for TransformerFG? The translations are mostly wrong.

A: That is expected. The purpose of Task 1 is to observe the warm-up behavior and study the architecture through the written questions, not to

produce a perfect translator. Report whatever translations you get, even if most are wrong.

Q: My visTransformer accuracy on CIFAR-10 is low.

A: Without hyperparameter tuning, 50–65% per-class accuracy is expected (see Slide 103). This is normal for a small from-scratch ViT on CIFAR-10.

Q: My visTransformer accuracy on the Food-101 subset is low.

A: With 20 classes and a small model at 64×64 resolution, 20–50% test accuracy is realistic. Random guessing gives 5%. As long as the loss goes down and accuracy is well above 5%, the attention visualizations will still show meaningful patterns. Report whatever accuracy you get.

Q: How do I extract attention weights without breaking the training code?

A: Store the weights as a `.detach()` attribute on the AttentionHead instance. This does not affect the computational graph or gradient flow. Only collect the weights at inference time (inside a `torch.no_grad()` block).

Q: The Food-101 download is very large.

A: Yes, approximately 5 GB for the full dataset (even though you only use 20 classes). Start the download early. If using Google Colab, download to your Google Drive to avoid re-downloading each session.

Q: Can I use a different pretrained ViT model for Task 4?

A: You may use any ViT variant from `timm` (e.g., `vit_small_patch16_224`, `deit_base_patch16_224`). Document which model you used. Do **not** use a non-ViT model (e.g., ResNet, EfficientNet) since the homework is about Transformers.

Q: Can I use different Food-101 classes than the 20 listed?

A: No. Use the exact 20 classes listed in Section 2.3 so that grading is consistent.

Q: Can I modify DLStudio code directly?

A: Copy the relevant classes from DLStudio's `Transformers` module into your own file and modify them there. Do **not** modify the installed DLStudio package. Cite the source wherever you reuse code.

Q: Can I work in a group?

A: No. This is individual work.

Q: I don't have a GPU.

A: Tasks 1 and 2 run on CPU (slowly, but they work). For Tasks 3 and 4, a GPU is strongly recommended. Use Google Colab's free GPU if needed.

References

[1] Avinash Kak, *DLStudio Platform*. Available at: https://engineering.purdue.edu/kak/distDLS/

[2] Avinash Kak, *Transformers: Learning with Purely Attention Based Networks*. Available at: https://engineering.purdue.edu/kak/pdf-kak/Transformers.pdf

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin, *Attention Is All You Need*, NeurIPS, 2017. Available at: https://arxiv.org/abs/1706.03762

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby, *An Image is Worth 16×16 Words: Transformers for Image Recognition at Scale*, ICLR, 2021. Available at: https://arxiv.org/abs/2010.11929

[5] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool, *Food-101 – Mining Discriminative Components with Random Forests*, European Conference on Computer Vision (ECCV), 2014. More details at: https://data.vision.ee.ethz.ch/cvl/datasets_extra/food-101/static/bossard_eccv14_food-101.pdf