

Deep Network Development

Assignments and Coding Exams Memorandum

Course Instructors

Tamás Takács, Imre Molnár

Course Demonstrators

Arpan Ekka, Máté Badó Miklós

Course Supervisor

Kristian Fenech

Department of Artificial Intelligence

Faculty of Informatics, ELTE

2026

Preface

This volume is the course memorandum for *Deep Network Development* (course code IPM-22fmiDNDEG), a Master's-level course offered by the Department of Artificial Intelligence, Faculty of Informatics, Eötvös Loránd University (ELTE), under the academic supervision of Dr. Kristian Fenech.

Scope of the course

Deep Network Development introduces students to the full landscape of modern deep learning through a weekly lecture paired with a hands-on practical session. This memorandum focuses on the practical side of the course. The semester covers the foundations of neural networks (linear and logistic regression, multi-layer perceptrons, backpropagation, optimizers), convolutional architectures for image classification and transfer learning, the four classical computer-vision tasks (object detection, semantic and instance segmentation, pose estimation) together with their modern deep-learning solutions, recurrent networks and attention mechanisms for sequence modelling, the Transformer architecture with its vision and language variants, and finally generative models and inverse problems such as depth estimation, optical flow, and text-guided image inpainting.

Structure of this memorandum

The memorandum is organized into three parts that mirror the weekly flow of the course:

Part I. Assignments. The two graded homework projects (image colorization, simplified object detection) and the large end-of-semester assignment (image captioning) that students complete over the course of the term.

Part II. Entry Examinations. Seven Jupyter-notebook entry examinations with reference solutions. These are the short hands-on assessments used to gate students into the final examination.

The graded midterm and final examinations, together with the quiz example collections, are reproduced in the companion volume *Deep Network Development Lecture Memorandum* that accompanies the lecture slide decks.

Authorship and attribution

The course content collected here was co-developed by Tamás Takács and Imre Molnár on an approximately equal basis. Two practice chapters, NMT with Transformers and Text-Guided Image Inpainting, were authored by Máté Badó Miklós and are attributed to him in their respective chapter headers.

The editorial work required to convert the Colab notebooks and exam documents into a single cohesive typographic volume was produced specifically for this grant submission.

Contents

Preface	2
I. Assignments	8
1. Homework 1: Image Colorization	9
1.1. Task Description	9
1.2. Expectations	9
1.3. Requirements	10
1.4. General Rules	11
1.5. Guidelines	11
1.6. 0. Necessary Imports	11
1.7. 1. Data Loading Process	11
1.8. 2. Defining Augmentations	12
1.9. 3. Creating Datasets and Dataloaders	12
1.10. 4.1 Visualize Training Data	13
1.11. 4.2 Creating a CNN Model	13
1.12. 4.3 Creating a FCN Model	13
1.13. 5.1 Plotting Model Parameter Count and Size	14
1.14. 5.2 Defining Loss Function and Optimizer	14
1.14.1. a. Mean Squared Error (MSE) Loss	14
1.14.2. b. Root Mean Squared Error (RMSE) Loss	15
1.14.3. c. Huber Loss	15
1.14.4. d. Structural Similarity Index (SSIM) Loss (Advanced)	15
1.15. 6. Training the Image Colorization Models	16
1.16. 7.1 Visualizing Training Metrics	16
1.17. 7.2 Running Inference on the Image Colorization Models	16
1.18. 8.1 Comparing Test Results	16
1.19. 8.2 Explaining the Difference in Performance	17
1.20. 9.1 Visualizing Feature Maps in the Convolutional Neural Network	17
1.21. 9.2 Visualizing Weights in the Fully Connected Network	17
2. Homework 2: Simplified Object Detection	19
2.1. Task Description	19
2.2. Expectations	19
2.3. Requirements	20
2.4. General Rules	21
2.5. Guidelines	21

2.6.	0. Necessary Imports	21
2.7.	1.1 Object Loading Process	22
2.8.	1.2 Crawling the Web for Background Images	22
2.9.	2. Creating a Synthetic Dataset	23
2.10.	3. Creating Dataloaders	24
2.11.	4. Visualizing the Training Data	24
2.12.	5. Creating a Custom Object Detection Model	24
2.13.	6.1 Plotting Model Parameter Count and Size	25
2.14.	6.2 Defining Loss Function and Optimizer	25
2.14.1.	a. Mean Squared Error (MSE) Loss (Bounding Box Regression)	25
2.14.2.	b. Cross-Entropy Loss (Classification Loss)	26
2.14.3.	c. Huber Loss (Smooth Bounding Box Regression Loss)	26
2.14.4.	d. Intersection over Union (IoU) Loss (Bounding Box Alignment Loss)	26
2.15.	7. Training the Custom Object Detection Model	27
2.16.	8.1 Visualizing Training Metrics	27
2.17.	8.2 Running Inference on the Custom Object Detection Model	27
2.17.1.	Key Object Detection Metrics	28
2.17.2.	Additional Model Performance Metrics	28
2.18.	8.3 Visualizing Model Predictions	28
2.19.	9. Loading an Existing Object Detection Model (YOLOvX)	28
2.20.	10. Evaluating the YOLO Model	29
3.	Assignment 1: Image Captioning	30
3.1.	Task Description	30
3.2.	Expectations	31
3.3.	Requirements	31
3.4.	General Rules	32
3.5.	Guidelines	33
3.6.	0. Necessary Imports	33
3.7.	1. Data Loading Process	33
3.8.	2. Defining Augmentations	34
3.9.	3. Creating Datasets and Dataloaders	34
3.10.	4.1 Visualizing Training Data	35
3.11.	4.2 Creating the Image Captioning Model	35
3.11.1.	Encoder-Attention-Decoder Architecture	35
3.12.	5. Defining Loss Function and Optimizer	36
3.12.1.	a. Cross-Entropy Loss (Standard Sequence Prediction Loss)	37
3.12.2.	b. CIDER Loss (Reinforcement Learning-Based Caption Quality Loss)	37
3.12.3.	c. Reinforcement Learning-Based Loss (REINFORCE with Self-Critical Sequence Training - SCST)	37
3.13.	6. Training the Image Captioning Model	38
3.14.	7.1 Visualizing Training Metrics	38
3.15.	7.2 Visualizing Attention Weights	39

3.16.	7.3 Running Inference on the Image Captioning Model	39
3.17.	8. Loading an Existing Image Captioning Model	39
3.18.	9. Evaluating the Existing Image Captioning Model	39
3.19.	10. Comparing the Two Models	40
II.	Entry Examinations	41
4.	Entry Exam 2025-05-23, Variant 1	42
4.1.	Student Information	42
4.2.	Overview	42
4.3.	Task Requirements	42
4.4.	Rules and Resources	42
4.5.	Submission Guidelines	43
4.6.	Retake Policy	43
4.7.	Task Description	43
4.8.	0. Necessary Imports and Data Loading	43
4.9.	1. Architecture	45
4.9.1.	Diagram Preview:	46
5.	Entry Exam 2025-05-23, Variant 2	49
5.1.	Student Information	49
5.2.	Overview	49
5.3.	Task Requirements	49
5.4.	Rules and Resources	49
5.5.	Submission Guidelines	50
5.6.	Retake Policy	50
5.7.	Task Description	50
5.8.	0. Necessary Imports and Data Loading	50
5.9.	1. Architecture	51
5.9.1.	Diagram Preview:	51
6.	Entry Exam 2025-06-06, Variant 1	54
6.1.	Student Information	54
6.2.	Overview	54
6.3.	Task Requirements	54
6.4.	Rules and Resources	54
6.5.	Submission Guidelines	55
6.6.	Retake Policy	55
6.7.	Task Description	55
6.8.	0. Necessary Imports and Data Loading	55
6.9.	1. Architecture	57
6.9.1.	Diagram Preview:	57

7. Entry Exam 2025-06-06, Variant 2	61
7.1. Student Information	61
7.2. Overview	61
7.3. Task Requirements	61
7.4. Rules and Resources	61
7.5. Submission Guidelines	62
7.6. Retake Policy	62
7.7. Task Description	62
7.8. 0. Necessary Imports and Data Loading	62
7.9. 1. Architecture	64
7.9.1. Diagram Preview:	64
8. Entry Exam 2025-06-17, Variant 1	68
8.1. Student Information	68
8.2. Overview	68
8.3. Task Requirements	68
8.4. Rules and Resources	68
8.5. Submission Guidelines	69
8.6. Retake Policy	69
8.7. Task Description	69
8.8. 0. Necessary Imports and Data Loading	69
8.9. 1. Architecture	74
8.9.1. Diagram Preview:	74
9. Entry Exam 2025-06-17, Variant 2	77
9.1. Student Information	77
9.2. Overview	77
9.3. Task Requirements	77
9.4. Rules and Resources	77
9.5. Submission Guidelines	78
9.6. Retake Policy	78
9.7. Task Description	78
9.8. 0. Necessary Imports and Data Loading	78
9.9. 1. Architecture	82
9.9.1. Diagram Preview:	82
10. Entry Exam 2025-06-30, Variant 1	85
10.1. Student Information	85
10.2. Overview	85
10.3. Task Requirements	85
10.4. Rules and Resources	85
10.5. Submission Guidelines	86
10.6. Retake Policy	86
10.7. Task Description	86

10.8.	0. Necessary Imports and Data Loading	86
10.9.	1. Architecture	90
10.9.1.	Diagram Preview:	90

Part I.

Assignments

Chapter 1

Homework 1: Image Colorization

Authors: Tamás Takács, Imre Molnár



2023.05.19 Wooden house Chapiali Salihorsk District.jpg
https://commons.wikimedia.org/wiki/File:2023.05.19_Wooden_house_Chapiali_Salihorsk_District.jpg
Tess Matthew, CC BY-SA 4.0 <<https://creativecommons.org/licenses/by-sa/4.0/>>, via Wikimedia Commons

This notebook contains the required task for the **first homework** of the **Deep Network Development (DNDEG)** course. Read the task description carefully and **fill in the empty code cells**.

1.1. Task Description

Image Colorization is the process of converting a grayscale image (single-channel) into an image with a specific color spectrum. In this homework, your task is to **create and train a convolutional neural network (CNN) model** to colorize grayscale images to the RGB color spectrum. To demonstrate the effectiveness of CNNs, you will also implement a **linear regression model** for the same task and compare their results.

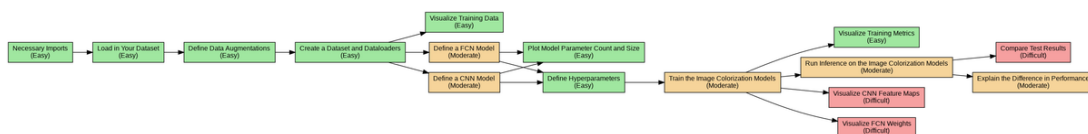
1.2. Expectations

- The **performance does not need to be perfect**, but your **metrics should show better results than random color guessing**, and the **loss should decrease throughout the epochs**.
- You are **free to design any custom shallow convolutional neural network**, as long as it is specifically tailored for the image colorization task.
- You can choose **any dataset** (e.g., CIFAR-10, Caltech101, Food101, Flickr) or create a custom dataset by **scraping the web** for a specific theme. If you decide to use web scraping, you can utilize the `icrawler` Python package. Ensure that **no NSFW images** are included.
- If you decide to use a **high-resolution dataset**, apply **resizing transformations** since the fully connected network's complexity increases with input size. For example:
 - A $[1 \times 32 \times 32]$ grayscale image will have 1,024 features.

- A $[1 \times 512 \times 512]$ grayscale image will have 262,144 features, which is **256 times more**.

1.3. Requirements

- Data Visualization:
 - Display **samples from the original dataset**.
 - Show the **transformed grayscale images** at the beginning of the notebook.
 - Present **predicted colorized images** alongside ground truth images.
- Performance Monitoring:
 - **Visualize training and validation losses** after training the network.
 - **Use plots and metrics (e.g., matplotlib, seaborn)** to display performance trends.
- Implement **overfitting prevention techniques**, such as:
 - **Early stopping**
 - **Regularization techniques**
 - **Experimenting with different loss functions**
 - Save the **best-performing model** during training.
- Compare model performance:
 - **Visualize predictions** on a batch of test images.
 - Calculate and display **batch losses for both FCN and CNN models**.
 - Explain why the models perform as they do, **highlighting the strengths and weaknesses** of each approach.
- Visualizing:
 - **Feature Maps** of the CNN model
 - **Weights** of the FCN model
- Optionally, use **WandB (Weights and Biases)** or **TensorBoard** to:
 - Track and visualize **training progress**.
 - Monitor **model performance** over time.
 - Log **key metrics**.



Each section of the Notebook will guide you through the task: - necessary imports - data loading process - defining data augmentations - creating a dataset and dataloaders - visualizing the training data - creating a CNN model - creating a FCN model - defining a loss function and an optimizer - creating a training and validation loop - testing the models - comparing results - visualizing weights and feature maps

The sections are there to guide you but you **do not have to follow them strictly**.

Copy this notebook to your drive (File -> Save a copy in Drive), edit it, and upload the final .ipynb file to [Canvas](#). If you are using this in **Google Colab**, save it as .ipynb and upload it, or share the public

link. If you have your own machine with Jupyter installed, you can work there as well, as long as you save and upload the .ipynb file.

1.4. General Rules

Please check all the requirements listed on **Canvas**, but here are some general rules:

- **No NSFW, inappropriate, offensive, or negative images.** Be creative or use your hobbies as inspiration.
- The model is **not expected to achieve high performance** but must perform better than random color guessing.
- Copying others' code will make you fail the assignment automatically, resulting in a **0**.
- Not submitting anything results in a **0**.
- Submitting something, as long as it is not an empty notebook, might result in a **1**.
- **Deadline is October 21st Tuesday 11:59 PM** (strict, no late submission)
- Feel free to add more code cells as needed. But don't put code into external Python files.
- Please add your name and Neptun ID below.

Good luck!

1.5. Guidelines

Please carefully read each cell of the notebook, as they contain guidelines to help you complete the assignments. While you don't have to follow them strictly, we believe that they provide enough help.

Name:

Neptun ID:

1.6. 0. Necessary Imports

Import all the necessary packages for this assignment. **ONLY PYTORCH MODELS ARE ACCEPTED!**

```
import os
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
from torchsummary import summary
from torch.utils.data import DataLoader, Dataset
from torchvision import transforms, datasets

device = torch.device("cuda:0") if torch.cuda.is_available() else torch.device("cpu")
```

1.7. 1. Data Loading Process

If you're unsure which dataset to use, **PyTorch** provides a collection of **pre-built datasets** in its [documentation](#). Look for a dataset under the *Image Classification* section that contains colored images.

Caltech-101:

[1]

Food-101:

1.8. 2. Defining Augmentations

Since most datasets and images are in RGB color space, you'll need to convert them to grayscale. However, we **strongly recommend** against applying the grayscale transformation immediately after loading the dataset, as this would result in losing the original colored images (which serve as the target). Instead, apply essential preprocessing steps like tensor conversion, normalization and resizing first, and **only perform the grayscale transformation within the custom Dataset class** to retain access to both input (grayscale) and target (RGB) images.

```
train_transforms = transforms.Compose([
    # Add Augmentations
])

test_transforms = transforms.Compose([
    # Add Augmentations
])
```

1.9. 3. Creating Datasets and Dataloaders

To train a custom object detection model, a dataset is required. For this assignment, you will create a **custom dataset class** that takes a dataset path as input and returns **input-output** image pairs when batched through a DataLoader:

- **Input:** $[1 \times H \times W]$ grayscale image
- **Output:** $[3 \times H \times W]$ RGB image

Use the built-in **transforms.Grayscale** method from `torchvision.transforms` to convert images to grayscale.

Note: The W and H dimensions do not necessarily have to be the same.

For the **DataLoader**, ensure that the **batch size** is appropriate so that it fits into memory. Set the **shuffle** parameter as follows:

- **Training & Validation DataLoaders:** `shuffle=True` (to randomize the order of samples)
- **Test DataLoader:** `shuffle=False` (to maintain consistency in evaluation)

```
class GrayscaleToColorDataset(Dataset):
    def __init__(self):
        raise NotImplementedError

    def __len__(self):
        raise NotImplementedError

    def __getitem__(self, idx):
        raise NotImplementedError
```


1.10. 4.1 Visualize Training Data

To visualize the training data, extract a batch from the training DataLoader and plot the **input-output** pairs using Matplotlib or Seaborn. Ensure that at least **8 pairs** are displayed for a clear representation.

```
def visualize_batch():
    raise NotImplementedError
```

1.11. 4.2 Creating a CNN Model

You are free to design your CNN using any combination of: - Conv2d layers - MaxPool2d - Normalization techniques (e.g., BatchNorm, Dropout) - Activation functions (e.g., ReLU, LeakyReLU, etc.)

The final convolutional layer should output a tensor of shape $[3 \times H \times W]$ to match the RGB color space. Apply a sigmoid activation at the end to constrain the output values between **0** and **1**, allowing for later **denormalization** to the full RGB scale.

For an **extra grade improvement**, explore more advanced architectures and techniques:

- **Dilation in Conv2D layers:** Expands the receptive field without increasing parameters.
[PyTorch Docs on Conv2D](#)
- **Transpose Convolutions:** Used for upsampling to restore spatial dimensions in an image.
[PyTorch Docs on ConvTranspose2d](#)
- **Encoder-Decoder Models & UNets:** Popular in segmentation and image-to-image translation tasks.
[Understanding UNet](#)
- **Generative Adversarial Networks (GANs):** Used for high-quality image generation and colorization.
[Introduction to GANs](#)

```
class ColorizationCNN(nn.Module):
    def __init__(self):
        super(ColorizationCNN, self).__init__()
        raise NotImplementedError

    def forward(self, x):
        raise NotImplementedError
```

1.12. 4.3 Creating a FCN Model

The fully connected network should process an **input grayscale image** and transform it into an RGB image by following these steps:

1. **Flatten the input:**
 - Convert the image into a vector of shape $[N, 1 * H * W]$, where N is the batch size.
2. **Linear Projection:**
 - Apply a **fully connected (linear) layer** that projects the input vector to a space of $[N, 3 * H * W]$.
3. **Reshape the output:**

- Reshape the result back into an image format with shape $[N, 3, H, W]$, matching the expected RGB output.

Feel free to add **more hidden layers** and incorporate **non-linear activation functions**, normalization techniques, and other enhancements to help the model learn more expressive and complex features.

```
class ColorizationLinear(nn.Module):
    def __init__(self):
        super(ColorizationLinear, self).__init__()
        raise NotImplementedError

    def forward(self, x):
        raise NotImplementedError
```

1.13. 5.1 Plotting Model Parameter Count and Size

A convenient way to inspect the number of parameters in a defined `nn.Module` is by using the `torchsummary` package. This package provides a detailed summary of the model architecture, including the number of parameters in each layer, their shapes, and the total parameter count.

To use `torchsummary`, you need to pass a dummy tensor with the specified input shape. The package will then traverse the model, layer by layer, calculating the total number of trainable and non-trainable parameters while also displaying the output shapes at each stage.

Summnarize **both models!**

```
model_cnn = ColorizationCNN().to(device)
summary(model_cnn, input_size = )
```

1.14. 5.2 Defining Loss Function and Optimizer

Loss Functions and their options:

For our grayscale-to-color image translation task, we are performing **regression** rather than classification. Our network predicts continuous RGB values for each pixel, meaning that the chosen loss function should effectively measure the difference between the predicted and true color values.

1.14.1. a. Mean Squared Error (MSE) Loss

MSE loss is a standard choice for regression tasks as it penalizes larger errors more strongly than smaller ones. It is computed as:

$$\mathcal{L}_{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

where y_i is the ground truth pixel value and $\hat{y}_i$ is the predicted pixel value.

- **Pros:** Simple, commonly used for regression, computationally efficient.
- **Cons:** Sensitive to outliers since it squares errors.

1.14.2. b. Root Mean Squared Error (RMSE) Loss

RMSE is simply the square root of MSE, reducing the impact of squared errors and providing a more interpretable loss value in the same unit as the output.

$$\mathcal{L}_{RMSE} = \sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2}$$

- **Pros:** Less sensitive to large errors compared to MSE, more interpretable.
 - **Cons:** Still punishes large errors significantly.
-

1.14.3. c. Huber Loss

Huber Loss is a combination of MSE and Mean Absolute Error (MAE) that is more robust to outliers. It applies MSE for small errors and MAE for large ones:

$$\mathcal{L}_{Huber} = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2, & \text{if } |y_i - \hat{y}_i| \leq \delta \\ \delta(|y_i - \hat{y}_i| - \frac{1}{2}\delta), & \text{otherwise} \end{cases}$$

where δ is a threshold defining the transition from quadratic to linear loss.

- **Pros:** Robust to outliers while still penalizing significant errors.
 - **Cons:** Introduces an extra hyperparameter δ that must be tuned.
-

1.14.4. d. Structural Similarity Index (SSIM) Loss (Advanced)

While MSE and Huber loss work on a per-pixel basis, SSIM measures **perceptual similarity** between two images. It compares contrast, structure, and luminance.

$$\mathcal{L}_{SSIM} = 1 - SSIM(y, \hat{y})$$

where SSIM measures image similarity in a windowed manner.

- **Pros:** Better reflects human perception of image quality.
 - **Cons:** More computationally expensive.
-

[PyTorch Documentation](#)

Optimizers and their options:

There are some pre-built [Optimizers in PyTorch](#), they are sufficient in most cases, especially if their parameters are well set. The two most well-known are Adam (AdamW) and SGD, both of which originate from Gradient Descent, which we implemented earlier.

- Stochastic Gradient Descent ([SGD](#))
- ADaptive Moment optimizer ([ADAM](#))
- [A good general overview](#)

1.15. 6. Training the Image Colorization Models

When implementing the training loop, ensure the following key aspects are included:

- Set an **appropriate number of epochs** for model training, balancing between underfitting and overfitting.
- Maintain running losses throughout each epoch and compute both **training loss** and **validation loss** per epoch.
- Implement an **early stopping mechanism** to halt training if validation loss stops improving, preventing unnecessary overfitting.
- **Save the model at its best-performing epoch** based on validation loss, ensuring the best version is retained for inference.

1.16. 7.1 Visualizing Training Metrics

- **Restore the model's parameters** from the checkpoint where validation loss was lowest to use the most optimal version of the model.
- Use Matplotlib or Seaborn to plot the loss curves over epochs.

Did your model **converge**? Explain your results!

```
def plot_losses():
    raise NotImplementedError
```

Answer:

1.17. 7.2 Running Inference on the Image Colorization Models

Pass test images through both the **CNN-based model** and the **Fully Connected Network (FCN) model**. Compute the test losses for both models to quantify performance.

```
def test_model():
    raise NotImplementedError
```

1.18. 8.1 Comparing Test Results

Select **two different test images** and generate comparison plots. - For each test image, create a **2-row, 3-column layout**: - Row 1 (CNN Model):

- Grayscale Input - CNN Colorized Prediction - Ground Truth RGB Image
- Row 2 (FCN Model):
- Grayscale Input - FCN Colorized Prediction - Ground Truth RGB Image

Display the loss value below each predicted image.

```
def plot_predictions():  
    raise NotImplementedError
```

1.19. 8.2 Explaining the Difference in Performance

Examine the differences in both the predicted test images and the computed test losses, providing reasoning for the performance variations between the two models. Provide explanations on how these architectural differences impact both qualitative (visual) and quantitative (loss-based) performance.

Answer:

1.20. 9.1 Visualizing Feature Maps in the Convolutional Neural Network

Your goal is to **visualize the feature maps** inside your trained convolutional neural network (CNN). Feature maps show how the network transforms an input image layer by layer.

1. Create a Hook Function:

- Define a function that will store the feature maps from a given layer during a forward pass.

2. Register Hooks to Convolutional Layers:

- Attach this function to **all convolutional layers** in your CNN model to capture their outputs.

3. Pass a Sample Image Through the Model:

- Select a grayscale image from the test dataset and feed it into the trained CNN.

4. Visualize the Feature Maps:

- For each convolutional layer, **plot its feature maps** (at most **10 feature maps per layer**. If a layer has fewer, visualize all of them).
- Include the **layer name** in each plot for clarity.

[Hook Functions](#)

1.21. 9.2 Visualizing Weights in the Fully Connected Network

In this task, you will explore what the first layer of your **fully connected neural network (FCN)** learns when colorizing grayscale images. Since the first linear layer directly maps grayscale pixels to hidden features, we can **reshape and visualize its weights** to understand what patterns the model is detecting.

1. **Extract the weights** of the first fully connected layer (e.g., `fc1`) from the trained linear model.
2. **Select a few neurons** (e.g., 5-10) and reshape their weights into 32×32 images.
3. **Plot these weights as grayscale images** using Matplotlib with a chosen colormap.

Each neuron in e.g. `fc1` has **1024 weights** (since $32 \times 32 = 1024$), meaning **each neuron “sees” the entire grayscale image**. By reshaping these **1024 weights** back into 32×32 , we get an image representation of **what each neuron is learning**. If the model has learned useful features, you should see structured patterns, like **edges or textures**.

The [weight variable](#)

Please make sure to download your `.ipynb` file, and upload it to **Canvas** on time!

Chapter 2

Homework 2: Simplified Object Detection

Authors: Tamás Takács, Imre Molnár

This notebook contains the required task for the **second homework** of the **Deep Network Development (DNDEG)** course. Read the task description carefully and **fill in the empty code cells**.

2.1. Task Description

You are required to **create and train a custom object detection model** to solve a simplified object detection task. Additionally, you will implement the same task using a **pre-trained YOLO model**.

To achieve this, you will first **generate a synthetic dataset** by placing a specific object onto a set of background images. This dataset will be used for training both models. Since this is a **simplified object detection problem**, each image should contain **only a single object**, ensuring a controlled training environment.

Your goal is to **train a custom object detection model from scratch** and later **fine-tune a pre-trained YOLO model** using the same dataset.

2.2. Expectations

- While **high performance is not expected**, your model should **perform better than random guessing**, and the **loss should consistently decrease across epochs**.
- You should structure your dataset using the **YOLO format for bounding box regression** (), where:
 - “ is the object class label (an integer starting from 0).
 - and are the normalized coordinates (values between 0 and 1) of the bounding box center.
 - and are the normalized dimensions (values between 0 and 1) of the bounding box relative to the image size.

ensuring that the same dataset can be seamlessly used for fine-tuning a **pre-trained YOLO model**.

- Your evaluation should include key metrics such as **precision, recall, F1-score, mAP (mean average precision), and IoU (Intersection over Union)**, along with monitoring the overall loss trends throughout training. These metrics should be **visualized using plots and graphs** (e.g., with **matplotlib** or **seaborn**) to illustrate performance over time.
- To prevent overfitting, you should implement techniques such as **Early Stopping** and ensure that the **best-performing model is saved** during training.

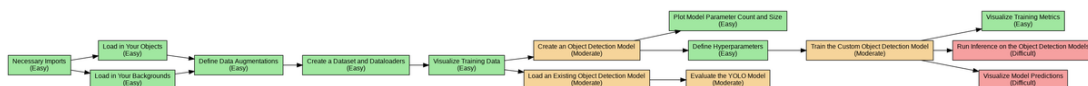
- For better experiment tracking and visualization, consider using **WandB (Weights and Biases)** or **TensorBoard** to log training progress, performance metrics, and model improvements in real time.

2.3. Requirements

You will be required to choose at least three different objects that you would like to run object detection on. For the background images, you are free to use anything thematic and download images utilizing a web crawler. A simple example could be Waldo, Wilma, and Wenda, with the background images being *Where's Waldo?* doodles.

- Data Visualization:
 - Display **your chosen objects with the background removed**.
 - Display **a few of your background images**.
 - Visualize the **synthetic dataset**, where each image should contain a background with an object placed onto it, along with a bounding box and class label.
- Performance Monitoring:
 - **Visualize training and validation losses** (classification loss, regression loss) after training the network.
 - Use **plots and metrics** (e.g., matplotlib, seaborn) to display performance trends.
- Implement overfitting prevention techniques, such as:
 - **Early stopping**
 - **Regularization techniques** (e.g., dropout, weight decay)
 - **Experimenting with different loss functions**
 - Save the **best-performing model** during training.
- Evaluate model performance:
 - **Visualize predictions** on a batch of test images.
 - Calculate **precision, recall, F1-score, mAP, and IoU** for the test dataset.
- Fine-Tuning YOLO:
 - Fine-tune a **YOLO nano model (version 5 or later)** on your object detection task.
 - Evaluate model performance using the **same metrics** as before.
- Optionally, use WandB (Weights and Biases) or TensorBoard to:
 - Track and visualize **training progress**.
 - Monitor **model performance** over time.
 - Log **key metrics**.

NOTE: A **multi-shot object detector is unnecessary** since you are dealing with **single-object detection**, for which a simple bounding box regression and classification head will suffice. Implementing Non-Maximum Suppression is also **not required** due to the single-object constraint. If you're feeling adventurous, you could extend this homework to multi-object detection using a single-shot approach for a **+1 improvement to your grade!**



Each section of the Notebook will guide you through the task: - necessary imports - loading in chosen objects - crawling the web for background images - creating a synthetic dataset - creating dataloaders - visualizing the training data - creating a custom model for object detection - defining a loss function and an optimizer - training and validating the custom model - testing the custom model - fine-tuning a YOLO model - evaluating YOLO results

The sections are there to guide you but you **do not have to follow them strictly**.

Copy this notebook to your drive (File -> Save a copy in Drive), edit it, and upload the final .ipynb file to [Canvas](#). If you are using this in **Google Colab**, save it as .ipynb and upload it, or share the public link. If you have your own machine with Jupyter installed, you can work there as well, as long as you save and upload the .ipynb file.

2.4. General Rules

Please check all the requirements listed on **Canvas**, but here are some general rules:

- **No NSFW, inappropriate, offensive, or negative images.** Be creative or use your hobbies as inspiration.
- The model is **not expected to achieve high performance** but must perform better than random guessing.
- Copying others' code will make you fail the assignment automatically, resulting in a **0**.
- Not submitting anything results in a **0**.
- Submitting something, as long as it is not an empty notebook, might result in a **1**.
- **Deadline is November 4th Tuesday 11:59 PM** (strict, no late submission)
- Feel free to add more code cells as needed. But don't put code into external Python files.
- Please add your name and Neptun ID below.

Good luck!

2.5. Guidelines

Please carefully read each cell of the notebook, as they contain guidelines to help you complete the assignments. While you don't have to follow them strictly, we believe that they provide enough help.

Name:

Neptun ID:

2.6. 0. Necessary Imports

Import all the necessary packages for this assignment. **ONLY PYTORCH MODELS ARE ACCEPTED!**

```
import torch
import os
import numpy as np
import matplotlib.pyplot as plt
```

```
import torchvision.transforms as transforms
from torch import nn
from torch.utils.data import Dataset
from torch.utils.data import DataLoader

device = torch.device("cuda:0") if torch.cuda.is_available() else torch.device("cpu")
```

2.7. 1.1 Object Loading Process

For this assignment you might choose 3 of the [Where's Wally Characters](#).

Please select 3 or more objects. After selecting the objects, **remove the background, keeping only the object itself**. You can do this manually, online, or by using code (this step is not part of the assignment evaluation).

As an example, the following objects were chosen (with the background removed using the Segment Anything Model). **Make sure to visualize all of your objects** with Matplotlib or Seaborn!

2.8. 1.2 Crawling the Web for Background Images

To train a custom object detection model, a dataset is required. For this assignment, you will create a **synthetic dataset** by:

- **Placing a single object at a random position** on a background image.

The goal is to follow a similar approach to **Where's Waldo doodles** by embedding objects into cluttered backgrounds. **Therefore, gather a selection of background images from various doodles**. This can be done manually or programmatically.

To automate image collection, it is recommended to use icrawler. [Documentation](#)

```
!pip install icrawler
```

```
!mkdir background
```

In case of any **errors, try using a different crawler**. You can use one of the following options:

“python from icrawler.builtin import BaiduImageCrawler, BingImageCrawler, GoogleImageCrawler

```
from icrawler.builtin import GoogleImageCrawler

google_crawler = GoogleImageCrawler(
    parser_threads=2,
    downloader_threads=4,
    storage={'root_dir': 'background'}
)

for keyword in ['doodle background', 'cluttered doodle background', 'colorful doodle background', 'waldo doodle background']:
    google_crawler.crawl(
        keyword=keyword, max_num=100, file_idx_offset='auto')

# Some images may return an error, but icrawler tries to find other images regardless
```

Make sure to **visualize a small subset of the background images as well!**

```
def display_sample_images(directory):
    raise NotImplementedError
```

2.9. 2. Creating a Synthetic Dataset

This is a crucial part of the assignment evaluation. Using the selected objects and collected background images, follow these steps to generate your dataset:

1. **Randomly select background images** from your dataset.
2. **Place a single object** from your selected set into each background image at a **random location**.
3. **Record the object's coordinates** and create a corresponding bounding box. The format should be:
 - If an object with height h and width w is positioned at (x, y) , the bounding box should follow:

$$[x_{\text{center}}, y_{\text{center}}, \text{width}, \text{height}]$$

- Normalize the values using:

$$\left[\frac{x + w/2}{\text{image width}}, \frac{y + h/2}{\text{image height}}, \frac{w}{\text{image width}}, \frac{h}{\text{image height}} \right]$$

where *image* refers to the background image dimensions.

4. **Save the generated dataset**, as it will be used for training both your custom model and fine-tuning an existing object detection model.
5. **Wrap the dataset creation** within a PyTorch Dataset class for seamless data handling and model training.
6. **Split the dataset** into training, validation, and test sets to ensure proper model evaluation.
7. **Apply various transformations** on the training and validation datasets. Only apply **tensor conversion and normalization** on the test set.

```
def createDataset(root_dir, dataset_name, split, img_size, num_images):
    #This my own function... It creates and saves a dataset in the following format:
    # root/
    #   dataset/
    #       train/
    #           images/
    #               00001.jpg
    #           labels/
    #               00001.txt with the content [class yolo_x, yolo_y, yolo_w, yolo_h] following the yolo format
    #       val
    # ...
    # ADD YOUR CODE HERE
    raise NotImplementedError
```

```

createDataset('dataset', 'background', split='train', img_size=(640,640), num_images=5000)
createDataset('dataset', 'background', split='val', img_size=(640,640), num_images=1000)
createDataset('dataset', 'background', split='test', img_size=(640,640), num_images=200)

class ObjectDetectionDataset(Dataset):
    def __init__(self, root_dir, split, num_classes, transform):
        raise NotImplementedError
    def __len__(self):
        raise NotImplementedError
    def __getitem__(self, idx):
        raise NotImplementedError

```

2.10. 3. Creating Dataloaders

For the **DataLoader**, ensure that the **batch size** is appropriate so that it fits into memory. Set the **shuffle** parameter as follows:

- **Training & Validation DataLoaders:** `shuffle=True` (to randomize the order of samples)
- **Test DataLoader:** `shuffle=False` (to maintain consistency in evaluation)

2.11. 4. Visualizing the Training Data

To visualize the training data, extract a batch from the training DataLoader and plot the **input images with their respective bounding box and class label** using Matplotlib or Seaborn. Ensure that at least **8 datapoints** are displayed for a clear representation.

```

def visualize_batch():
    raise NotImplementedError

```

2.12. 5. Creating a Custom Object Detection Model

In this assignment, you are tasked with building your own custom object detection model.

Design a CNN-based architecture that includes a **backbone for feature extraction** and **two output branches**:

- One for **class probabilities** (with output size equal to the number of object classes, e.g., 3).
- Another for **bounding box regression** (with size 4 for the box coordinates: $[x_{center}, y_{center}, width, height]$).

This is the **core evaluation of the assignment**. Experiment with different layers, hyperparameters, and regularization techniques to optimize the model's performance.

Note: You **ARE** allowed to use a pre-trained backbone (VGG, ResNet) for this task. You are free to fine-tune the pre-trained backbone or just freeze is completely.

```

class CustomModel(nn.Module):
    def __init__(self, num_classes=3):
        super(CustomModel, self).__init__()
        raise NotImplementedError

```

```
def forward(self, x):
    raise NotImplementedError
```

2.13. 6.1 Plotting Model Parameter Count and Size

A convenient way to inspect the number of parameters in a defined `nn.Module` is by using the `torchsummary` package. This package provides a detailed summary of the model architecture, including the number of parameters in each layer, their shapes, and the total parameter count.

To use `torchsummary`, you need to pass a dummy tensor with the specified input shape. The package will then traverse the model, layer by layer, calculating the total number of trainable and non-trainable parameters while also displaying the output shapes at each stage.

Summnmarize **both models!**

```
model = CustomModel().to(device)
summary(model, input_size = )
```

2.14. 6.2 Defining Loss Function and Optimizer

Loss Functions and their options:

For our custom object detection task, we are performing both **regression** (for bounding box coordinates) and **classification** (for object categories). Our network predicts **continuous** bounding box values, along with a **discrete** class label, meaning that our chosen loss function should effectively be a composite of two losses:

1. A **regression loss** for bounding box prediction.
2. A **classification loss** for object label prediction.

Here are some common loss functions that can be used:

2.14.1. a. Mean Squared Error (MSE) Loss (Bounding Box Regression)

MSE loss is a standard choice for regression tasks, as it penalizes larger errors more strongly than smaller ones. For bounding box prediction, this ensures that predicted box coordinates are as close as possible to the ground truth.

$$\mathcal{L}_{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

where y_i is the ground truth bounding box coordinate and $\hat{y}_i$ is the predicted coordinate.

- **Pros:** Simple, computationally efficient, commonly used in regression tasks.
- **Cons:** Sensitive to outliers since it squares errors, which may cause instability when predicting bounding boxes.

2.14.2. b. Cross-Entropy Loss (Classification Loss)

For predicting object classes, **Cross-Entropy Loss** is the most commonly used loss function in classification tasks. It is defined as:

$$\mathcal{L}_{CE} = - \sum_{i=1}^C y_i \log(\hat{y}_i)$$

where \$ C \$ is the total number of object classes, \$ y_i \$ is the ground truth label (one-hot encoded), and \$ \hat{y}_i \$ is the predicted probability for that class.

- **Pros:** Standard for classification, provides well-calibrated probabilities.
- **Cons:** Can be affected by class imbalances. Label smoothing or weighted loss may be needed.

2.14.3. c. Huber Loss (Smooth Bounding Box Regression Loss)

Huber Loss is an improvement over MSE that reduces sensitivity to outliers. It applies MSE for small errors and Mean Absolute Error (MAE) for larger errors:

$$\mathcal{L}_{Huber} = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2, & \text{if } |y_i - \hat{y}_i| \leq \delta \\ \delta(|y_i - \hat{y}_i| - \frac{1}{2}\delta), & \text{otherwise} \end{cases}$$

where \$ \delta \$ is a threshold that determines when the loss transitions from quadratic to linear.

- **Pros:** More robust than MSE, reduces the effect of outliers on bounding box predictions.
- **Cons:** Requires tuning of \$ \delta \$ for optimal performance.

2.14.4. d. Intersection over Union (IoU) Loss (Bounding Box Alignment Loss)

IoU Loss directly optimizes the overlap between the predicted and ground-truth bounding boxes:

$$\mathcal{L}_{IoU} = 1 - \frac{\text{Intersection Area}}{\text{Union Area}}$$

This loss ensures that the model prioritizes bounding box alignment rather than just minimizing coordinate differences.

- **Pros:** More appropriate for object detection since it directly optimizes box overlap.
- **Cons:** Harder to optimize, as gradients may vanish when boxes do not overlap.

[PyTorch Documentation](#)

Optimizers and their options:

There are some pre-built [Optimizers in PyTorch](#), they are sufficient in most cases, especially if their parameters are well set. The two most well-known are Adam (AdamW) and SGD, both of which originate from Gradient Descent, which we implemented earlier.

- **Stochastic Gradient Descent** ([SGD](#))
- **ADAPtive Moment optimizer** ([ADAM](#))
- [A good general overview](#)

2.15. 7. Training the Custom Object Detection Model

When implementing the training loop, ensure the following key aspects are included:

- Set an **appropriate number of epochs** for model training, balancing between underfitting and overfitting.
- Maintain running losses throughout each epoch and compute both **training losses** and **validation losses** per epoch.
- Implement an **early stopping mechanism** to halt training if validation loss stops improving, preventing unnecessary overfitting.
- **Save the model at its best-performing epoch** based on validation loss, ensuring the best version is retained for inference.
- Ensure you're optimizing both **classification loss** and **bounding box regression loss**.

2.16. 8.1 Visualizing Training Metrics

- **Restore the model's parameters** from the checkpoint where validation loss was lowest to use the most optimal version of the model.
- Use Matplotlib or Seaborn to plot the loss curves over epochs.

Did your model **converge**? Explain your results!

```
def plot_losses():  
    raise NotImplementedError
```

Answer:

2.17. 8.2 Running Inference on the Custom Object Detection Model

After training your **custom object detection model**, evaluate its performance by passing test images through it. Select **at least three** key evaluation metrics to assess the model's effectiveness. Potential metrics include:

2.17.1. Key Object Detection Metrics

- **Precision:** Measures the proportion of correctly detected objects among all predicted objects.
- **Recall:** Represents the proportion of correctly detected objects out of all actual objects present in the image.
- **F1-Score:** The harmonic mean of Precision and Recall, balancing false positives and false negatives.
- **IoU (Intersection over Union):** Computes the ratio of overlap between the predicted bounding box and the ground truth bounding box.
- **mAP (mean Average Precision):** A widely used metric that calculates precision at multiple recall thresholds and averages them across all object classes.

2.17.2. Additional Model Performance Metrics

- **Inference Speed:** Measure the time taken to process a single image.
- **Number of Parameters:** Count the total trainable parameters in your model.
- **Model Size:** Evaluate the storage space required for the trained model.

```
def test_model()
    raise NotImplementedError

def calculate_iou()
    raise NotImplementedError

def calculate_map()
    raise NotImplementedError
```

2.18. 8.3 Visualizing Model Predictions

- Select a set of test images and display them with **predicted bounding boxes** overlaid.
- Compare predictions against the **ground truth bounding boxes** to visually assess model accuracy.
- Annotate each prediction with relevant metrics (Precision, Recall, IoU) to highlight how well the model performs.

For detailed tracking and logging, consider using **Weights and Biases (WandB)** or **TensorBoard** to visualize training progress, compare models, and gain deeper insights into your results.

2.19. 9. Loading an Existing Object Detection Model (YOLOvX)

You are free to select any YOLO model version available. We recommend referring to the **Ultralytics documentation** on object detection: [Ultralytics YOLO Docs](#).

Training a YOLO model is straightforward and requires a properly formatted **data.yaml** file, which defines dataset paths and classes. Refer to the **official GitHub repository** for detailed instructions: [Ultralytics GitHub](#).

Once your dataset is prepared, you can initiate training directly from the command line using a simple command:

```
!yolo detect train data=datasets/data.yaml model=yolov11n.pt epochs=100 imgsz=640
```


A data.yaml might look something like this:

```
with open('dataset/data.yaml', 'w') as f:
    f.write("path: \ntrain: content/dataset/train/images \nval: content/dataset/val/images \ntest: content/dataset/test/images\n")
    f.write("nc: 3 \n")
    f.write("names:\n 0: wilma\n 1: waldo\n 2: wenda\n")
    f.write('\n')
f.close()
```

2.20. 10. Evaluating the YOLO Model

Ensure that you **log and visualize training losses** and **evaluation metrics** throughout the training process. YOLO models typically provide verbose output by default, but you should also generate visualizations to track performance trends effectively.

Please make sure to download your .ipynb file, and upload it to **Canvas** on time!

```
from datetime import datetime, timedelta, timezone

deadline = datetime(2025, 11, 4, 23, 59, 0, tzinfo=timezone.utc)

def time_until_deadline():
    now = datetime.now(timezone.utc)
    remaining = deadline - now
    if remaining.total_seconds() <= 0:
        return "Time's up!"
    days = remaining.days
    hours, remainder = divmod(remaining.seconds, 3600)
    minutes, _ = divmod(remainder, 60)

    return f"{days} days, {hours} hours, {minutes} minutes"

print("Time left until submission:", time_until_deadline())
```

Time left until submission: 13 days, 13 hours, 48 minutes

Chapter 3

Assignment 1: Image Captioning

Authors: Tamás Takács, Imre Molnár

A tabby cat with blue eyes looks at the camera. 	A brown dog stands outdoors in profile. 	A corded white dog stands on a sandy beach. 	Zebras graze among trees on a savanna. 
An elephant walks across dry grassland. 	People walk through a station concourse. 	Cars and buses queue in heavy city traffic. 	A breakfast plate and coffee sit on a table. 

This notebook contains the required task for the **first assignment** of the **Deep Network Development (DNDEG)** course. Read the task description carefully and **fill in the empty code cells**.

3.1. Task Description

Your task is to train your **own custom image captioning model** and compare its performance with an existing **pre-trained** model. You will work with the Flickr8k [dataset](#), ensuring it is properly split into **training**, **validation**, and **test** sets. The model should follow an **Encoder-Decoder + Attention** structure, where the encoder can use a **pre-trained backbone** such as *ResNet*, but it must be **fine-tuned**. An **attention layer** is required to visualize what the network has learned. After training, you will **evaluate and compare** the performance of your model against a **pre-trained** one, analyzing the **generated captions** and the **attention mechanisms**.

The Flickr8k dataset consists of **8000 images**, each paired with **5 captions**, resulting in a total of **40,000 captions**. It is your **design choice** how you implement the **dataset and batching process**. A common approach is to treat each caption as a separate data point, yielding **40,000 datapoints** in total.

The **tokenization process** is also up to you. Since the dataset contains a vast vocabulary, it may include words that appear only once. To improve model efficiency, it is **good practice to limit the vocabulary size**.

During **validation and testing**, every image has **5 reference captions**. To ensure a robust **BLEU score calculation**, it is recommended to compute BLEU scores using **all 5 matching captions** as reference captions for each image.

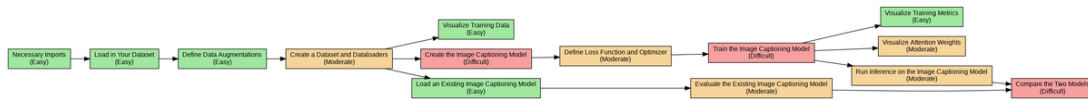
3.2. Expectations

- Your model is **not expected to achieve state-of-the-art performance**, but it should perform **better than random guessing**. The **loss should decrease** throughout training, and the model's performance should be monitored to prevent overfitting or underfitting.
- You will work with the Flickr8k dataset, which should be split into **train, validation, and test** sets. The training set is used to train the image captioning model, the validation set helps fine-tune hyperparameters and monitor performance, and the test set is used for final evaluation.
- To assess model performance, include **visualizations of loss values** and **evaluation metrics** for the dataset splits.
- You are free to choose any **pre-trained model** (blip, vit-gpt2-flickr8k, etc.) for model comparison at the end.

3.3. Requirements

- Data Preparation and Visualization:
 - Ensure the[2] **dataset** is correctly split into **train, validation, and test sets**.
 - Display **sample images with their original captions**.
 - Visualize **tokenized captions** to ensure proper preprocessing.
- Model Training and Performance Monitoring:
 - Train an **Encoder-Decoder + Attention** model using the training set.
 - Track **training and validation loss** using visualizations (e.g., Matplotlib, Seaborn).
 - Monitor **BLEU-1, BLEU-2, BLEU-3, and BLEU-4 scores** for a greedy decoder throughout training.
- Implement **techniques to avoid overfitting**, such as:
 - **Early stopping** to halt training when validation loss stops improving.
 - **Regularization** (e.g., dropout, weight decay) to improve generalization.
 - **Experimenting with different loss functions** to find the most effective approach.
 - **Saving the best-performing model** during training for later evaluation.
- Model Comparison:
 - Evaluate and compare your **custom-trained model** with a **pre-trained model** on the same batch of test images.
 - **Visualize generated captions** and compare them to ground-truth captions.
 - Compute **BLEU scores** using all five reference captions for a given image.
 - Analyze and explain the **strengths and weaknesses** of both models based on performance differences.
- Optionally, use **WandB (Weights and Biases)** or **TensorBoard** to:
 - Track and visualize **training progress**.

- Monitor **model performance** over time.
- Log **key metrics**.



Each section of the Notebook will guide you through the task:

- necessary imports
- data loading process
- defining data augmentations
- creating a dataset and dataloaders
- visualizing the training data
- creating a CNN model
- creating a FCN model
- defining a loss function and an optimizer
- creating a training and validation loop
- testing the models
- comparing results
- visualizing weights and feature maps

The sections are there to guide you but you **do not have to follow them strictly**.

Copy this notebook to your drive (File -> Save a copy in Drive), edit it, and upload the final .ipynb file to [Canvas](#). If you are using this in **Google Colab**, save it as .ipynb and upload it, or share the public link. If you have your own machine with Jupyter installed, you can work there as well, as long as you save and upload the .ipynb file.

3.4. General Rules

Please check all the requirements listed on **Canvas**, but here are some general rules:

- The model is **not expected to achieve high performance** but must perform better than random color guessing.
- Copying others' code will make you fail the assignment automatically, resulting in a **0**.
- Not submitting anything results in a **0**.
- Submitting something, as long as it is not an empty notebook, might result in a **1**.
- **Deadline is November 25th Tuesday 11:59 PM** (strict, no late submission)
- Feel free to add more code cells as needed. But don't put code into external Python files.
- Please add your name and Neptun ID below.

Good luck!

In order to get your grade for an assignment, you must defend it by answering questions during the designated practice time.

3.5. Guidelines

Please carefully read each cell of the notebook, as they contain guidelines to help you complete the assignments. While you don't have to follow them strictly, we believe that they provide enough help.

Name:

Neptun ID:

3.6. 0. Necessary Imports

Import all the necessary packages for this assignment. **ONLY PYTORCH MODELS ARE ACCEPTED!**

```
import os
import torch
import nltk
import torchvision
import torchaudio
import matplotlib.pyplot as plt
import random
from torch.utils.data import Dataset, DataLoader
from PIL import Image
from torchvision import transforms

device = torch.device("cuda:0") if torch.cuda.is_available() else torch.device("cpu")
```

3.7. 1. Data Loading Process

For this assignment you will be using the [Flickr8k](#) dataset, which contains **captions/descriptions** of different images.



Actual Captions :-

Three zebras graze among acacia trees .
A herd of zebras stands in dry grassland .
Zebras feeding on grass in an open savanna .
Several zebras in a field with scattered trees .
A group of zebras grazes near trees in the Serengeti .

Predicted Caption : a zebra standing in the middle of a field .

The best way to download this dataset is through Kaggle. First **create a token**, download it and upload it here. Follow [these](#) steps.

3.8. 2. Defining Augmentations

When applying **augmentations** to the Flickr8k dataset, it is important to note that these transformations should be applied **only to the images** and not to the captions.

Ensure that your **data augmentation pipeline** includes: - A **normalization step** to scale pixel values appropriately. - A **tensor conversion step** to transform images into tensors for model compatibility. - Additional **augmentations of your choice**, such as random cropping, flipping, or color jittering, to enhance model generalization.

```
train_transforms = transforms.Compose([
    # Add Augmentations
])

test_transforms = transforms.Compose([
    # Add Augmentations
])
```

3.9. 3. Creating Datasets and Dataloaders

To load the **Flickr8k** dataset, you need to create a **custom PyTorch Dataset** class that returns **images and their corresponding captions**. The captions should be **tokenized** before being returned.

Make sure to include special tokens in your tokenized captions: - sos (Start of Sentence) - eos (End of Sentence) - unk (Unknown Token) for words outside the vocabulary.

It is recommended to build a **Vocabulary class** to store all the words in your dataset, as your model can only generate words that exist in this vocabulary. However, saving every word is unnecessary. A **common practice** is to include only words that appear **at least 5 times** across the entire dataset to reduce noise and improve model efficiency.

For the **DataLoader**, ensure that the **batch size** is appropriate so that it fits into memory. Set the **shuffle** parameter as follows:

- **Training & Validation DataLoaders:** shuffle=True (to randomize the order of samples)
- **Test DataLoader:** shuffle=False (to maintain consistency in evaluation)

Note: Remember that, a batch has **different examples**, and each example (caption) might have **different sizes**. Therefore, consider padding techniques for the captions.

```
class FlickrDataset(Dataset):
    def __init__(self,):
        raise NotImplementedError

    def __len__(self):
        raise NotImplementedError

    def __getitem__(self, idx):
        raise NotImplementedError
```

3.10. 4.1 Visualizing Training Data

To visualize the training data, extract a batch from the training `DataLoader` and plot the **input-target** pairs using `Matplotlib` or `Seaborn`. Ensure that at least **8 pairs** are displayed for a clear representation.

Make sure to **visualize the original and the tokenized caption** as well!

```
def visualize_batch():  
    raise NotImplementedError
```

3.11. 4.2 Creating the Image Captioning Model

For this assignment, **you are required to create your own custom image captioning model and compare its performance** with an existing pre-trained model.

3.11.1. Encoder-Attention-Decoder Architecture

Your model should follow an **Encoder-Attention-Decoder** structure with the following components:

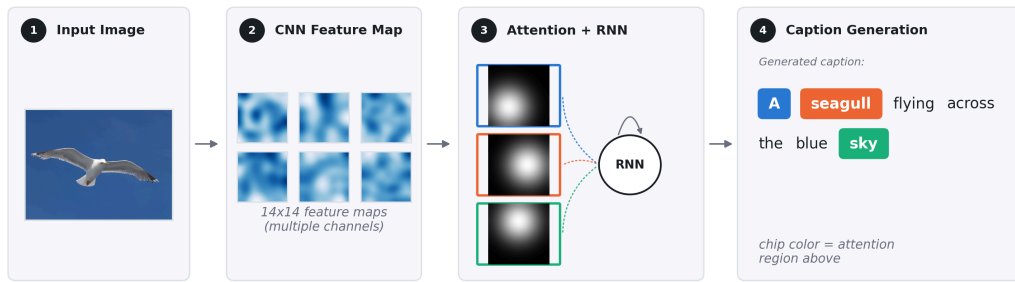
- The **Encoder** processes images to extract meaningful features. You can either design a **custom convolutional architecture** or use a **pre-trained model** like *ResNet*, but **fine-tuning is required** to adapt the features to the captioning task.
 - The **Attention Layer** acts as an interface between the encoder and decoder. It leverages the extracted image features to compute **attention scores**, helping the decoder focus on relevant parts of the image during caption generation. For simplicity, use **Linear layers** to implement the attention mechanism.
 - The **Decoder** is a sequence-based model (e.g., *LSTM*) that processes the image features and generates captions sequentially.
-

Extra Credit Opportunity

- Additional points will be awarded for implementing either a **Vision Transformer** [\[\[3\]\]](#) (ViT) **encoder** or a **Transformer-based decoder** instead of an LSTM.

Here is an example of something similar to what we want to create:

Image Captioning with Visual Attention



Seagull flying (5).jpg [https://commons.wikimedia.org/wiki/File:Seagull_flying_\(5\).jpg](https://commons.wikimedia.org/wiki/File:Seagull_flying_(5).jpg) Arnold Paul, CC BY-SA 2.5
<<https://creativecommons.org/licenses/by-sa/2.5/>>, via Wikimedia Commons

```
class Encoder(nn.Module):
    def __init__(self):
        super(Encoder, self).__init__()
        return NotImplementedError

    def forward(self, images):
        return NotImplementedError

class Attention(nn.Module):
    def __init__(self, encoder_dim, decoder_dim, attention_dim):
        super(Attention, self).__init__()
        return NotImplementedError

    def forward(self, features, hidden_state):
        return NotImplementedError

class Decoder(nn.Module):
    def __init__(self, embed_size, vocab_size, attention_dim, encoder_dim, decoder_dim):
        return NotImplementedError

    def forward(self, features, captions):
        return NotImplementedError

class EncoderDecoder(nn.Module):
    def __init__(self, embed_size, vocab_size, attention_dim, encoder_dim, decoder_dim):
        return NotImplementedError

    def forward(self, images, captions):
        return NotImplementedError
```

3.12. 5. Defining Loss Function and Optimizer

Loss Functions and their options:

In an **image captioning task**, the model generates a sequence of words conditioned on an image. This involves both **sequence generation (language modeling)** and **image understanding**, making loss selection crucial for effective training. The primary loss function should optimize **word prediction** while ensuring **grammatical correctness and semantic coherence**.

Since our task involves **predicting discrete word tokens**, we typically use **sequence-based loss functions**, but we can also explore loss functions that account for **semantic meaning and alignment**.

3.12.1. a. Cross-Entropy Loss (Standard Sequence Prediction Loss)

The most common loss for text generation tasks is **Cross-Entropy Loss**, which measures the difference between the **predicted word probability distribution** and the **true word** in the sequence. It is computed as:

$$\mathcal{L}_{CE} = - \sum_{t=1}^T y_t \log(\hat{y}_t)$$

where: - T is the sequence length (number of words in the caption), - y_t is the **ground truth word** (one-hot encoded), - $\hat{y}_t$ is the **predicted probability** of that word.

- **Pros:** Simple, well-established for text generation tasks, easy to implement.
- **Cons:** Treats each word prediction independently, ignoring sentence-level meaning, but useful during teacher forcing.

3.12.2. b. CIDER Loss (Reinforcement Learning-Based Caption Quality Loss)

Cross-Entropy loss focuses on **token-level** accuracy, but it does not capture **sentence-level fluency and meaning**. CIDER (Consensus-based Image Description Evaluation) helps optimize captions towards human-like descriptions.

$$\mathcal{L}_{CIDEr} = 1 - CIDEr(\hat{Y}, Y)$$

where $CIDEr(\hat{Y}, Y)$ measures how similar the generated caption $\hat{Y}$ is to multiple reference captions Y .

- **Pros:** Optimizes for caption similarity to human references, improves fluency. Best suited for **fine-tuning after pretraining with Cross-Entropy Loss**.
- **Cons:** Harder to optimize, requires reinforcement learning.

3.12.3. c. Reinforcement Learning-Based Loss (REINFORCE with Self-Critical Sequence Training - SCST)

Since captioning is **sequential**, we can treat it as a reinforcement learning problem. Instead of directly predicting words, we train the model to maximize **rewards** (e.g., BLEU or CIDEr scores). This is done using the **REINFORCE** algorithm:

$$\mathcal{L}_{RL} = -(r(\hat{Y}) - r(\bar{Y})) \sum_{t=1}^T \log P(y_t | y_{1:t-1}, X)$$

where: $-r(\hat{Y})$ is the reward for the generated caption, $-r(\tilde{Y})$ is the baseline reward (e.g., score from a greedy decoder).

- **Pros:** Optimizes **sentence-level metrics** instead of per-word accuracy, leading to better captions.
- **Cons:** Computationally expensive, requires careful tuning.

PyTorch Documentation

Optimizers and their options:

There are some pre-built [Optimizers in PyTorch](#), they are sufficient in most cases, especially if their parameters are well set. The two most well-known are Adam (AdamW) and SGD, both of which originate from Gradient Descent, which we implemented earlier.

- Stochastic Gradient Descent ([SGD](#))
- ADaptive Moment optimizer ([ADAM](#))
- [A good general overview](#)

3.13. 6. Training the Image Captioning Model

When implementing the training loop, ensure the following key aspects are included:

- Set an **appropriate number of epochs** for model training, balancing between underfitting and overfitting.
- Maintain running losses throughout each epoch and compute both **training loss** and **validation loss** per epoch.
- Implement an **early stopping mechanism** to halt training if validation loss stops improving, preventing unnecessary overfitting.
- **Save the model at its best-performing epoch** based on validation loss, ensuring the best version is retained for inference.

Note: Pay attention to your choices. **Be aware of overfitting** and underfitting. Make sure the **loss is decreasing over the epochs**. Save the losses so that they can be visualized.

3.14. 7.1 Visualizing Training Metrics

- **Restore the model's parameters** from the checkpoint where validation loss was lowest to use the most optimal version of the model.
- Use Matplotlib or Seaborn to plot the loss curves over epochs.

Did your model **converge**? Explain your results!

```
def plot_losses():
    raise NotImplementedError
```

3.15. 7.2 Visualizing Attention Weights

Visualize the attention weights in order to understand how your model learned. For that, use the context variable returned by the **Attention** class to overlay them on the image.

```
def plot_attention():  
    raise NotImplementedError
```

3.16. 7.3 Running Inference on the Image Captioning Model

Pass test images through the **custom-trained image captioning model**, then evaluate its performance on the test set. Use the **BLEU score** as the evaluation metric, implementing **BLEU-1**, **BLEU-2**, **BLEU-3**, and **BLEU-4**.

To improve performance on the validation and test sets, you may use **beam search decoding** instead of standard greedy decoding.

Beam Search is a more sophisticated decoding algorithm that **considers multiple possible caption sequences at each time step instead of selecting the most probable word at every step (greedy decoding)**. It maintains a fixed number of candidate sequences (beam width) and expands them based on their cumulative probabilities, ultimately selecting the **most likely complete caption**.

Using beam search can lead to **more fluent and accurate captions** by avoiding suboptimal word choices that greedy decoding might make.

```
def test_model()  
    raise NotImplementedError
```

3.17. 8. Loading an Existing Image Captioning Model

You are free to select any pre-trained image captioning model available. We recommend considering the following options:

1. **BLIP (Bootstrapping Language-Image Pre-training):** BLIP is a versatile vision-language pre-training framework that excels in both understanding and generation tasks. It effectively utilizes noisy web data by generating and filtering synthetic captions, achieving SOTA results in image captioning. Pre-trained models and code are available on [GitHub](#) and the [Hugging Face Model Hub](#).
2. **ViT-GPT2 Image Captioning Model:** This model combines a Vision Transformer (ViT) as the encoder and GPT-2 [\[\[4\]](#) as the decoder, effectively connecting visual inputs with text generation. A fine-tuned version on the Flickr8k dataset is accessible on the [Hugging Face Model Hub](#).

3.18. 9. Evaluating the Existing Image Captioning Model

For evaluation, apply the **same metrics** used in the assessment of your custom image captioning model to ensure a consistent and fair comparison.

3.19. 10. Comparing the Two Models

Compare the performance of both models using **BLEU-1, BLEU-2, BLEU-3, and BLEU-4** scores. Visualize predictions from both models on the **same batch of test images** to highlight their differences.

Analyze and justify the differences in performance, discussing factors such as model architecture, attention mechanisms, and training strategies. Finally, **propose possible improvements for your custom model**.

Answer:

Please make sure to download your .ipynb file, and upload it to **Canvas** on time!

```
from datetime import datetime, timedelta, timezone

deadline = datetime(2025, 11, 25, 23, 59, 0, tzinfo=timezone.utc)

def time_until_deadline():
    now = datetime.now(timezone.utc)
    remaining = deadline - now
    if remaining.total_seconds() <= 0:
        return "Time's up!"
    days = remaining.days
    hours, remainder = divmod(remaining.seconds, 3600)
    minutes, _ = divmod(remainder, 60)

    return f"{days} days, {hours} hours, {minutes} minutes"

print("Time left until submission:", time_until_deadline())
```

```
Time left until submission: 21 days, 14 hours, 3 minutes
```

Part II.

Entry Examinations

Chapter 4

Entry Exam 2025-05-23, Variant 1

4.1. Student Information

- **Name:** <Enter your full name>
 - **Neptun ID:** <Enter your Neptun ID>
 - **Date:** 23/05/2025

 - **Time Slot:** 09:00 - 09:30 (30 minutes)
-

4.2. Overview

This coding task must be completed to qualify for the second part of the final exam (1-hour written session). Final course grades will be assigned based on overall performance during the active semester, provided both components are passed.

4.3. Task Requirements

- Implement a **neural network architecture**, including a **functioning forward pass**.
 - The code must run without errors and meet the minimum implementation criteria.
-

4.4. Rules and Resources

Allowed	Not Allowed
Internet access	Communication with others (e.g., Teams, WhatsApp, Messenger)
AI tools (e.g., ChatGPT, Copilot)	Collaboration or assistance from peers
Practice notebooks	Use of group chats, forums, or messaging apps

Any violation will result in **immediate disqualification** and a **failing grade**.

4.5. Submission Guidelines

- Submit your solution as a **.ipynb file** on **Canvas**
 - Late submissions or incomplete solutions will not be accepted
 - Your code must be fully executable and meet the minimum task requirements
-

4.6. Retake Policy

- If this part is failed, **one retake opportunity** will be provided
 - Failing the retake results in a **fail for the course**
-

Good luck!

4.7. Task Description

Your task is to implement a **custom neural network architecture**, including a fully functional `forward()` method.

You will be provided with a **diagram of the model architecture**. Some components will contain a **?** **symbol**, indicating that you must determine and specify the appropriate hyperparameters or configurations based on the overall architecture and problem.

Please note the following: - **All layers shown in the diagram must be included** in your implementation. - In some cases, **connections may not be fully drawn**, but this does not mean the layer is optional. - Only the **final output dimensions** are required to match exactly. - The implementation will be **manually reviewed**, so correctness of structure and logic is essential.

4.8. 0. Necessary Imports and Data Loading

```
# Cell 0.1 (DO NOT EDIT THIS CELL!)
```

```
import os
import torch
import gdown
import requests
import torch.nn as nn
import matplotlib.pyplot as plt

from PIL import Image
from torchvision import transforms
```

```
# Cell 0.2 (DO NOT EDIT THIS CELL!)
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device
```

```
device(type='cuda')
```

```
# Cell 0.3 -> INPUT (DO NOT EDIT THIS CELL!)
url = 'https://drive.google.com/uc?id=1NUW-91kBqkx5bhiJDmtD_DyXvjFzo8sD'
output = 'input_image.png'

try:
    gdown.download(url, output, quiet=False)
    input_image = Image.open(output).convert('RGB')

    transform = transforms.Compose([
        transforms.Resize((128, 128)),
        transforms.ToTensor(),
    ])

    input_tensor = transform(input_image).unsqueeze(0).to(device)
except:
    input_tensor = torch.randn(1, 3, 128, 128)
    input_image = transforms.ToPILImage()(input_tensor.squeeze(0).to(device))
```

```
Downloading...
From: https://drive.google.com/uc?id=1NUW-91kBqkx5bhiJDmtD_DyXvjFzo8sD
To: /content/input_image.png
100%|#####| 224k/224k [00:00<00:00, 4.13MB/s]
```

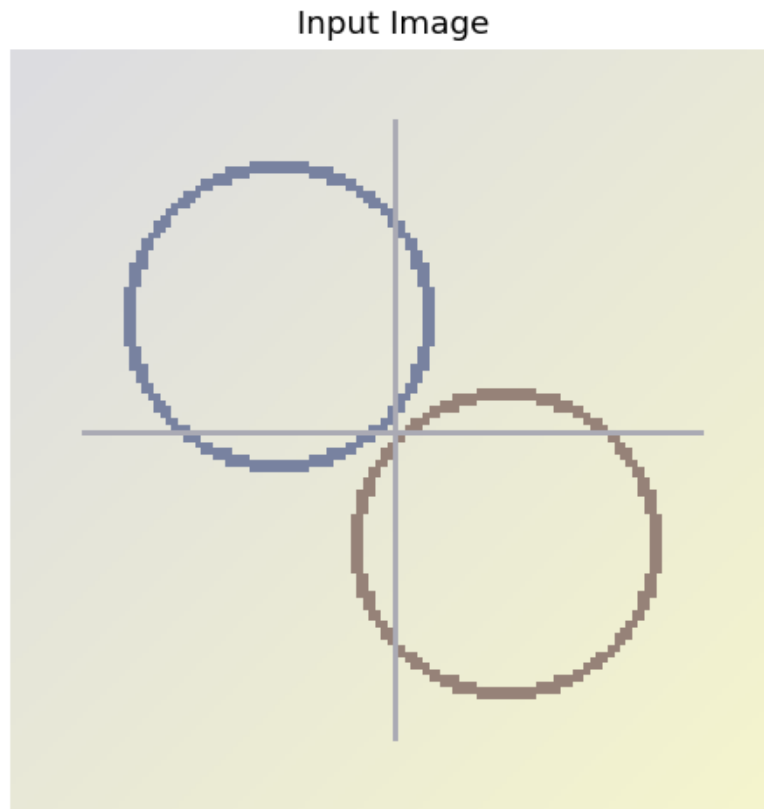
```
# Cell 0.4 -> INPUT (DO NOT EDIT THIS CELL!)
vocab_size = 16
input_tokens = torch.randint(0, vocab_size, (1, 16)).to(device)
print(input_tokens.shape)
```

```
torch.Size([1, 16])
```

```
# Cell 0.5 (DO NOT EDIT THIS CELL!)
print("Tensor shape:", input_tensor.shape)

plt.imshow(input_image)
plt.axis('off')
plt.title("Input Image")
plt.show()
```

```
Tensor shape: torch.Size([1, 3, 128, 128])
```

4.9. 1. Architecture

To better view the architecture diagram:

- **Right-click the image** and select **“Open image in a new tab”** to enable zoom for a clearer view.
- Alternatively, you can **download the image** using the link below:

[Download Architecture Diagram](#)


```

print(f"[x] After BatchNorm2d #2: {x.shape}")
x = self.relu_1(x)
print(f"[x] After ReLU #2: {x.shape}")

y = self.embedding(text)
print(f"[y] After Embedding: {y.shape}")
y_top = y

y = y.view(1, 16, 32, 32)
print(f"[y] After reshape to spatial: {y.shape}")

x = torch.cat((x, x_bottom, y), dim=1)
print(f"[x] After Concatenation: {x.shape}")

x = x.view(x.size(0), -1, x.size(3))
print(f"[x] After Reshape for LSTM: {x.shape}")

x, _ = self.lstm(x)
print(f"[x] After LSTM: {x.shape}")

y_top = self.linear(y_top)
print(f"[y_top] After Linear: {y_top.shape}")

noise_2 = torch.randn_like(x)
x = x + noise_2
print(f"[x] After Adding Second Noise: {x.shape}")

return x

```

Cell 1.2 (DO NOT EDIT THIS CELL!)

```

custom_model = CustomModel().to(device)

output = custom_model(input_tensor, input_tokens)

print("Output Shape:", output.shape)

try:
    assert output.shape == (1, 1536, 256), "Output shape is incorrect."
    print("\n Congratulations! Your implementation is correct. You passed the first part of the exam! ")
except AssertionError as e:
    print(f"\n Error: {e}")

```

```

[image] Input image shape: torch.Size([1, 3, 128, 128])
[text] Input text shape: torch.Size([1, 16])
[x_top] After MaxPool2d: torch.Size([1, 3, 64, 64])
[x_top] After ConvTranspose2d: torch.Size([1, 3, 128, 128])
[x] After adding noise and x_top: torch.Size([1, 3, 128, 128])
[x] After Conv2d #1: torch.Size([1, 16, 64, 64])
[x] After BatchNorm2d #1: torch.Size([1, 16, 64, 64])
[x] After ReLU #1: torch.Size([1, 16, 64, 64])
[x_bottom] After MaxPool2d: torch.Size([1, 16, 32, 32])
[x] After Conv2d #2: torch.Size([1, 16, 32, 32])
[x] After BatchNorm2d #2: torch.Size([1, 16, 32, 32])
[x] After ReLU #2: torch.Size([1, 16, 32, 32])
[y] After Embedding: torch.Size([1, 16, 1024])
[y] After reshape to spatial: torch.Size([1, 16, 32, 32])
[x] After Concatenation: torch.Size([1, 48, 32, 32])

```

```
[x] After Reshape for LSTM: torch.Size([1, 1536, 32])  
[x] After LSTM: torch.Size([1, 1536, 256])  
[y_top] After Linear: torch.Size([1, 16, 256])  
[x] After Adding Second Noise: torch.Size([1, 1536, 256])  
Output Shape: torch.Size([1, 1536, 256])
```

Congratulations! Your implementation is correct. You passed the first part of the exam!

Chapter 5

Entry Exam 2025-05-23, Variant 2

5.1. Student Information

- **Name:** <Enter your full name>
 - **Neptun ID:** <Enter your Neptun ID>
 - **Date:** 23/05/2025

 - **Time Slot:** 09:00 - 09:30 (30 minutes)
-

5.2. Overview

This coding task must be completed to qualify for the second part of the final exam (1-hour written session). Final course grades will be assigned based on overall performance during the active semester, provided both components are passed.

5.3. Task Requirements

- Implement a **neural network architecture**, including a **functioning forward pass**.
 - The code must run without errors and meet the minimum implementation criteria.
-

5.4. Rules and Resources

Allowed	Not Allowed
Internet access	Communication with others (e.g., Teams, WhatsApp, Messenger)
AI tools (e.g., ChatGPT, Copilot)	Collaboration or assistance from peers
Practice notebooks	Use of group chats, forums, or messaging apps

Any violation will result in **immediate disqualification** and a **failing grade**.

5.5. Submission Guidelines

- Submit your solution as a **.ipynb file** on **Canvas**
 - Late submissions or incomplete solutions will not be accepted
 - Your code must be fully executable and meet the minimum task requirements
-

5.6. Retake Policy

- If this part is failed, **one retake opportunity** will be provided
 - Failing the retake results in a **fail for the course**
-

Good luck!

5.7. Task Description

Your task is to implement a **custom neural network architecture**, including a fully functional `forward()` method.

You will be provided with a **diagram of the model architecture**. Some components will contain a **?** **symbol**, indicating that you must determine and specify the appropriate hyperparameters or configurations based on the overall architecture and problem.

Please note the following: - **All layers shown in the diagram must be included** in your implementation. - In some cases, **connections may not be fully drawn**, but this does not mean the layer is optional. - Only the **final output dimensions** are required to match exactly. - The implementation will be **manually reviewed**, so correctness of structure and logic is essential.

5.8. 0. Necessary Imports and Data Loading

```
# Cell 0.1 (DO NOT EDIT THIS CELL!)
```

```
import os
import torch
import gdown
import requests
import torch.nn as nn
import matplotlib.pyplot as plt

from PIL import Image
from torchvision import transforms
```

```
# Cell 0.2 (DO NOT EDIT THIS CELL!)
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device
```

```
device(type='cuda')
```

5.9. 1. Architecture

To better view the architecture diagram:

- **Right-click the image** and select **“Open image in a new tab”** to enable zoom for a clearer view.
- Alternatively, you can **download the image** using the link below:

[Download Architecture Diagram](#)

5.9.1. Diagram Preview:

```
# Cell 1.1 (EDIT THIS CELL!)

class CustomModel(nn.Module):
    def __init__(self, img_channels=3, lstm_hidden=128):
        super().__init__()
        # -- Encoder branches -----
        # top branch: Conv(3→16,k=5,s=3,p=1) → BN → MaxPool(k=2,s=2)
        self.conv1 = nn.Conv2d(img_channels, 16, kernel_size=5, stride=3, padding=1)
        self.bn1 = nn.BatchNorm2d(16)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        # bottom branch: Conv(3→16,k=3,s=2,p=1) → BN → MaxPool(k=3,s=3)
        self.conv2 = nn.Conv2d(img_channels, 16, kernel_size=3, stride=2, padding=1)
        self.bn2 = nn.BatchNorm2d(16)
        self.pool2 = nn.MaxPool2d(kernel_size=3, stride=3)

        # -- Gating block -----
        # After concat(16+16+1=33) → gate_conv → sigmoid → multiply → proc_conv
        self.gate_conv = nn.Conv2d(48, 32, kernel_size=5, stride=1, padding=2)
        self.sigmoid = nn.Sigmoid()
        self.proc_conv = nn.Conv2d(32, 16, kernel_size=5, stride=1, padding=2)

        # -- Head into LSTM -----
        self.fc_enc = nn.Linear(400, lstm_hidden)

        # -- Sequence model -----
        self.lstm = nn.LSTM(input_size=lstm_hidden,
                            hidden_size=lstm_hidden,
                            num_layers=2,
                            batch_first=True)

        # -- Decoder -----
        self.fc_dec = nn.Linear(lstm_hidden, 32 * 8 * 8)
        self.deconv = nn.ConvTranspose2d(
            in_channels=32,
            out_channels=32,
            kernel_size=3,
            stride=2,
            padding=1,
            output_padding=1
        )

    def forward(self, x):
```

```

# -- two-branch encoder -----
# Branch 1
conv1_out = self.conv1(x)
print(f"[conv1_out] after conv1: {conv1_out.shape}")
bn1_out = self.bn1(conv1_out)
print(f"[bn1_out] after bn1: {bn1_out.shape}")
pool1_out = self.pool1(bn1_out)
print(f"[pool1_out] after pool1: {pool1_out.shape}") # → (B,16,5,5)

# Branch 2
conv2_out = self.conv2(x)
print(f"[conv2_out] after conv2: {conv2_out.shape}")
bn2_out = self.bn2(conv2_out)
print(f"[bn2_out] after bn2: {bn2_out.shape}")
pool2_out = self.pool2(bn2_out)
print(f"[pool2_out] after pool2: {pool2_out.shape}") # → (B,16,5,5)

# gaussian noise channel
noise = torch.randn(
    pool1_out.shape,
    device=x.device, dtype=pool1_out.dtype
)
print(f"[noise] gaussian channel: {noise.shape}")

# concat: (16 + 16 + 1) → 33 channels
x_cat = torch.cat([pool1_out, pool2_out, noise], dim=1)
print(f"[x_cat] after concat: {x_cat.shape}") # → (B,33,5,5)

# -- Gating block -----
x_proj = self.gate_conv(x_cat)
print(f"[x_proj] after gate_conv: {x_proj.shape}")
gate = self.sigmoid(x_proj)
print(f"[gate] after sigmoid: {gate.shape}")
gated = x_proj * gate
print(f"[gated] after gating: {gated.shape}") # → (B,32,5,5)

refined = self.proc_conv(gated)
print(f"[refined] after proc_conv: {refined.shape}")
y = refined + pool1_out
print(f"[y] after skip+refine: {y.shape}") # → (B,32,5,5)

# -- into LSTM -----
y_flat = y.view(y.size(0), -1)
print(f"[y_flat] after flatten: {y_flat.shape}") # → (B,800)
enc = self.fc_enc(y_flat)
print(f"[enc] after fc_enc: {enc.shape}") # → (B,128)

seq = enc.unsqueeze(1)
print(f"[seq] for LSTM input: {seq.shape}") # → (B,1,128)
out, h = self.lstm(seq)
print(f"[out] after LSTM: {out.shape}")
h_last = out[:, -1, :]
print(f"[h_last] last LSTM state: {h_last.shape}") # → (B,128)

# -- decode -----
dec_lin = self.fc_dec(h_last)
print(f"[dec_lin] after fc_dec: {dec_lin.shape}") # → (B,2048)
dec_res = dec_lin.view(-1, 32, 8, 8)
print(f"[dec_res] to (B,32,8,8): {dec_res.shape}")
dec_deconv = self.deconv(dec_res)

```



```

print(f"[dec_deconv] after deconv: {dec_deconv.shape}")# → (B,32,16,16)
dec_f      = dec_deconv.view(dec_deconv.size(0), dec_deconv.size(1), -1)
print(f"[dec_f] final output:      {dec_f.shape}")      # → (B,32,256)

return dec_f

```

Cell 1.2 (DO NOT EDIT THIS CELL!)

```

custom_model = CustomModel().to(device)
img = torch.randn(1, 3, 32, 32).to(device)

output = custom_model(img)

print("Output Shape:", output.shape)

try:
    assert output.shape == (1, 32, 256), "Output shape is incorrect."
    print("\n Congratulations! Your implementation is correct. You passed the first part of the exam! ")
except AssertionError as e:
    print(f"\n Error: {e}")

```

```

[conv1_out] after conv1: torch.Size([1, 16, 10, 10])
[bn1_out] after bn1:    torch.Size([1, 16, 10, 10])
[pool1_out] after pool1: torch.Size([1, 16, 5, 5])
[conv2_out] after conv2: torch.Size([1, 16, 16, 16])
[bn2_out] after bn2:    torch.Size([1, 16, 16, 16])
[pool2_out] after pool2: torch.Size([1, 16, 5, 5])
[noise] gaussian channel: torch.Size([1, 16, 5, 5])
[x_cat] after concat: torch.Size([1, 48, 5, 5])
[x_proj] after gate_conv: torch.Size([1, 32, 5, 5])
[gate] after sigmoid:    torch.Size([1, 32, 5, 5])
[gated] after gating:    torch.Size([1, 32, 5, 5])
[refined] after proc_conv: torch.Size([1, 16, 5, 5])
[y] after skip+refine:    torch.Size([1, 16, 5, 5])
[y_flat] after flatten:  torch.Size([1, 400])
[enc] after fc_enc:      torch.Size([1, 128])
[seq] for LSTM input:    torch.Size([1, 1, 128])
[out] after LSTM:        torch.Size([1, 1, 128])
[h_last] last LSTM state: torch.Size([1, 128])
[dec_lin] after fc_dec:   torch.Size([1, 2048])
[dec_res] to (B,32,8,8): torch.Size([1, 32, 8, 8])
[dec_deconv] after deconv: torch.Size([1, 32, 16, 16])
[dec_f] final output:    torch.Size([1, 32, 256])
Output Shape: torch.Size([1, 32, 256])

```

Congratulations! Your implementation is correct. You passed the first part of the exam!

Chapter 6

Entry Exam 2025-06-06, Variant 1

6.1. Student Information

- **Name:** *<Enter your full name>*
 - **Neptun ID:** *<Enter your Neptun ID>*
 - **Date:** 23/05/2025

 - **Time Slot:** 09:00 - 09:30 (30 minutes)
-

6.2. Overview

This coding task must be completed to qualify for the second part of the final exam (1-hour written session). Final course grades will be assigned based on overall performance during the active semester, provided both components are passed.

6.3. Task Requirements

- Implement a **neural network architecture**, including a **functioning forward pass**.
 - The code must run without errors and meet the minimum implementation criteria.
-

6.4. Rules and Resources

Allowed	Not Allowed
Internet access	Communication with others (e.g., Teams, WhatsApp, Messenger)
AI tools (e.g., ChatGPT, Copilot)	Collaboration or assistance from peers
Practice notebooks	Use of group chats, forums, or messaging apps

Any violation will result in **immediate disqualification** and a **failing grade**.

6.5. Submission Guidelines

- Submit your solution as a **.ipynb file** on **Canvas**
 - Late submissions or incomplete solutions will not be accepted
 - Your code must be fully executable and meet the minimum task requirements
-

6.6. Retake Policy

- If this part is failed, **one retake opportunity** will be provided
 - Failing the retake results in a **fail for the course**
-

Good luck!

6.7. Task Description

Your task is to implement a **custom neural network architecture**, including a fully functional `forward()` method.

You will be provided with a **diagram of the model architecture**. Some components will contain a **?** **symbol**, indicating that you must determine and specify the appropriate hyperparameters or configurations based on the overall architecture and problem.

Please note the following: - **All layers shown in the diagram must be included** in your implementation. - If the shapes **do not match** check the set parameters, **interpolation and resize** do not considered a valid mitigation of the issue. - If you use **torch.nn.functional** it counts as a **layer or block**. - In some cases, **connections may not be fully drawn**, but this does not mean the layer is optional. - Only the **final output dimensions** are required to match exactly. - The implementation will be **manually reviewed**, so correctness of structure and logic is essential.

6.8. 0. Necessary Imports and Data Loading

```
# Cell 0.1 (DO NOT EDIT THIS CELL!)

import os
import torch
import gdown
import requests
import torch.nn as nn
import matplotlib.pyplot as plt

from PIL import Image
from torchvision import transforms
```

```
# Cell 0.2 (DO NOT EDIT THIS CELL!)
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device
```

```
device(type='cpu')
```

```
import torch
from torch.fx import symbolic_trace
import torch.nn as nn
import torch.nn.functional as F
from collections import Counter

# Map human-readable names to their corresponding types/functions
required_counts = {
    'Conv2d': 3,
    'ReLU': 2,
    'BatchNorm2d': 2,
    'MaxPool2d': 1,
    'Linear': 4,
    'ConvTranspose2d': 3,
    'MaxUnpool2d': 1,
    'cat': 1, # optional, set as needed
    'reshape': 2, # will match F.reshape, x.reshape(), nn.Flatten, nn.Unflatten,
}

# Mapping from module/function type to the key in required_counts
name_map = {
    nn.Conv2d: 'Conv2d',
    nn.ReLU: 'ReLU',
    nn.BatchNorm2d: 'BatchNorm2d',
    nn.MaxPool2d: 'MaxPool2d',
    nn.Flatten: 'reshape',
    nn.Linear: 'Linear',
    nn.ConvTranspose2d: 'ConvTranspose2d',
    nn.Unflatten: 'reshape',
    nn.MaxUnpool2d: 'MaxUnpool2d',
    torch.flatten: 'reshape',
    torch.cat: 'cat',
    torch.add: 'add',
    torch.reshape: 'reshape',
    F.relu: 'ReLU',
    F.batch_norm: 'BatchNorm2d', # in case batchnorm is done functionally
    F.max_pool2d: 'MaxPool2d',
    F.linear: 'Linear',
    F.conv2d: 'Conv2d',
    F.conv_transpose2d: 'ConvTranspose2d',
    F.max_unpool2d: 'MaxUnpool2d',
}

def count_fx_ops(model):
    traced = symbolic_trace(model)
    op_counter = Counter()

    for node in traced.graph.nodes:
        if node.op == 'call_module':
            mod = dict(traced.named_modules())[node.target]
            name = name_map.get(type(mod))
            if name:
                op_counter[name] += 1
```

```

elif node.op in {'call_function', 'call_method':
    name = name_map.get(node.target)
    if name:
        op_counter[name] += 1

return op_counter

def check_op_counts(actual_counts, required_counts):
    issues = []
    for key, required in required_counts.items():
        actual = actual_counts.get(key, 0)
        if actual != required:
            issues.append(f"{key}: expected {required}, found {actual}")
    return issues

```

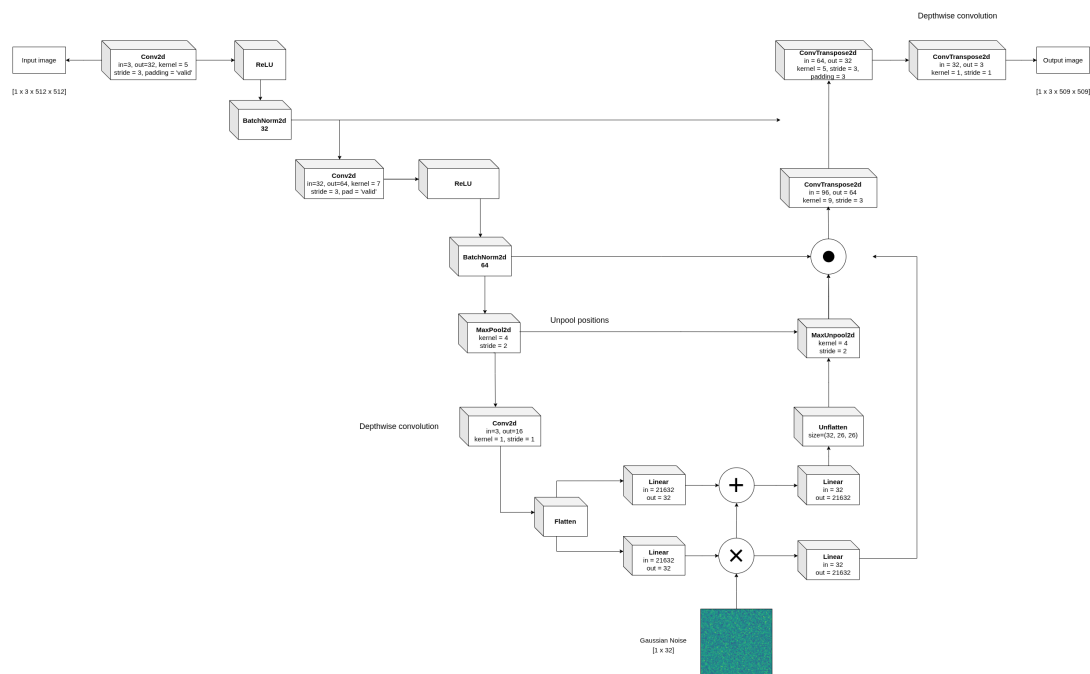
6.9. 1. Architecture

To better view the architecture diagram:

- **Right-click the image** and select “**Open image in a new tab**” to enable zoom for a clearer view.
- Alternatively, you can **download the image** using the link below:

[Download Architecture Diagram](#)

6.9.1. Diagram Preview:



Cell 1.1 (EDIT THIS CELL!)

```

class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        # ADD YOUR CODE HERE

```

```

self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=5, stride=3, padding=0)
self.relu1 = nn.ReLU()
self.batchnorm1 = nn.BatchNorm2d(32)

self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=7, stride=3, padding=0)
self.relu2 = nn.ReLU()
self.batchnorm2 = nn.BatchNorm2d(64)

# depthwise convolution
self.conv3 = nn.Conv2d(in_channels=64, out_channels=32, kernel_size=1, stride=1, padding=0)

self.maxpool1 = nn.MaxPool2d(kernel_size=4, stride=2, return_indices=True)

self.flatten = nn.Flatten()

self.mu = nn.Linear(21632, 32)
self.sigma = nn.Linear(21632, 32)

self.latent = nn.Linear(32, 21632)
self.latent2 = nn.Linear(32, 21632)

self.unflatten = nn.Unflatten(dim=1, unflattened_size=(32, 26, 26))

self.unpool = nn.MaxUnpool2d(kernel_size=4, stride=2)

self.unconv1 = nn.ConvTranspose2d(in_channels=96, out_channels=64, kernel_size=9, stride=3, padding=0)

self.unconv2 = nn.ConvTranspose2d(in_channels=64, out_channels=32, kernel_size=5, stride=3, padding=3)

# depthwise convolution
self.unconv3 = nn.ConvTranspose2d(in_channels=32, out_channels=3, kernel_size=1, stride=1, padding=0)

def forward(self, x):
    print(f"[image] Input image shape: {x.shape}")

    # ADD YOUR CODE HERE

    x = self.conv1(x)
    x = self.relu1(x)
    c1 = self.batchnorm1(x)
    print(f"[conv1] Output shape: {c1.shape}")

    x = self.conv2(c1)
    x = self.relu2(x)
    c2 = self.batchnorm2(x)
    print(f"[conv2] Output shape: {c2.shape}")

    x = self.conv3(c2)
    print(f"[conv3] Output shape: {x.shape}")
    x_shape_before_pooling = x.shape

    pool1, indices = self.maxpool1(x)
    print(f"[maxpool1] Output shape: {pool1.shape}")

    x = self.flatten(pool1)
    print(f"[flatten] Output shape: {x.shape}")

    mu = self.mu(x)
    sigma = self.sigma(x)
    print(f"[mu] Output shape: {mu.shape}")

```

```

print(f"[sigma] Output shape: {sigma.shape}")

eps = torch.randn_like(sigma)
z = mu + sigma * eps
print(f"[latent] Output shape: {z.shape}")

x = self.latent(z)
print(f"[latent2] Output shape: {x.shape}")
_ = self.latent2(z)

x = self.unflatten(x)
print(f"[unflatten] Output shape: {x.shape}")

x = self.unpool(x, indices, output_size=x_shape_before_pooling)
print(f"[unpool] Output shape: {x.shape}")

print(f'[before cat] {x.shape} {c2.shape}')
x = torch.cat((x, c2), dim=1)
print(f'[after cat] {x.shape}')
c_t = self.unconv1(x)
print(f"[unconv1] Output shape: {x.shape}")

x = self.unconv2(c_t)
print(f"[unconv2] Output shape: {x.shape}")

x = self.unconv3(x)

return x

```

Cell 1.2 (DO NOT EDIT THIS CELL!)

```

custom_model = CustomModel().to(device)
img = torch.randn(1, 3, 512, 512).to(device)

output = custom_model(img)

print("Output Shape:", output.shape)

# TESTING

expected = (1, 3, 509, 509)
if output.shape == expected:
    print("\n Output shape is correct.\n")
else:
    print(" Output shape is incorrect. Expected: {output.shape}, Actual: {output.shape}")

actual_counts = count_fx_ops(custom_model)
issues = check_op_counts(actual_counts, required_counts)

if issues:
    print("\n Mismatch found:")
    for issue in issues:
        print(" -", issue)
else:
    print("\n Model seems to satisfies all module count requirements.")

```

```

[image] Input image shape: torch.Size([1, 3, 512, 512])
[conv1] Output shape: torch.Size([1, 32, 170, 170])
[conv2] Output shape: torch.Size([1, 64, 55, 55])
[conv3] Output shape: torch.Size([1, 32, 55, 55])

```

```
[maxpool1] Output shape: torch.Size([1, 32, 26, 26])
[flatten] Output shape: torch.Size([1, 21632])
[mu] Output shape: torch.Size([1, 32])
[sigma] Output shape: torch.Size([1, 32])
[latent] Output shape: torch.Size([1, 32])
[latent2] Output shape: torch.Size([1, 21632])
[unflatten] Output shape: torch.Size([1, 32, 26, 26])
[unpool] Output shape: torch.Size([1, 32, 55, 55])
[before cat] torch.Size([1, 32, 55, 55]) torch.Size([1, 64, 55, 55])
[after cat] torch.Size([1, 96, 55, 55])
[unconv1] Output shape: torch.Size([1, 96, 55, 55])
[unconv2] Output shape: torch.Size([1, 32, 509, 509])
Output Shape: torch.Size([1, 3, 509, 509])
```

Output shape is correct.

```
[image] Input image shape: Proxy(getattr_1)
[conv1] Output shape: Proxy(getattr_2)
[conv2] Output shape: Proxy(getattr_3)
[conv3] Output shape: Proxy(getattr_4)
[maxpool1] Output shape: Proxy(getattr_5)
[flatten] Output shape: Proxy(getattr_6)
[mu] Output shape: Proxy(getattr_7)
[sigma] Output shape: Proxy(getattr_8)
[latent] Output shape: Proxy(getattr_9)
[latent2] Output shape: Proxy(getattr_10)
[unflatten] Output shape: Proxy(getattr_11)
[unpool] Output shape: Proxy(getattr_13)
[before cat] Proxy(getattr_14) Proxy(getattr_15)
[after cat] Proxy(getattr_16)
[unconv1] Output shape: Proxy(getattr_17)
[unconv2] Output shape: Proxy(getattr_18)
```

Model seems to satisfies all module count requirements.

Chapter 7

Entry Exam 2025-06-06, Variant 2

7.1. Student Information

- **Name:** <Enter your full name>
 - **Neptun ID:** <Enter your Neptun ID>
 - **Date:** 23/05/2025

 - **Time Slot:** 09:00 - 09:30 (30 minutes)
-

7.2. Overview

This coding task must be completed to qualify for the second part of the final exam (1-hour written session). Final course grades will be assigned based on overall performance during the active semester, provided both components are passed.

7.3. Task Requirements

- Implement a **neural network architecture**, including a **functioning forward pass**.
 - The code must run without errors and meet the minimum implementation criteria.
-

7.4. Rules and Resources

Allowed	Not Allowed
Internet access	Communication with others (e.g., Teams, WhatsApp, Messenger)
AI tools (e.g., ChatGPT, Copilot)	Collaboration or assistance from peers
Practice notebooks	Use of group chats, forums, or messaging apps

Any violation will result in **immediate disqualification** and a **failing grade**.

7.5. Submission Guidelines

- Submit your solution as a **.ipynb file** on **Canvas**
 - Late submissions or incomplete solutions will not be accepted
 - Your code must be fully executable and meet the minimum task requirements
-

7.6. Retake Policy

- If this part is failed, **one retake opportunity** will be provided
 - Failing the retake results in a **fail for the course**
-

Good luck!

7.7. Task Description

Your task is to implement a **custom neural network architecture**, including a fully functional `forward()` method.

You will be provided with a **diagram of the model architecture**. Some components will contain a **?** **symbol**, indicating that you must determine and specify the appropriate hyperparameters or configurations based on the overall architecture and problem.

Please note the following: - **All layers shown in the diagram must be included** in your implementation. - If the shapes **do not match** check the set parameters, **interpolation and resize** do not considered a valid mitigation of the issue. - If you use **torch.nn.functional** it counts as a **layer or block**. - In some cases, **connections may not be fully drawn**, but this does not mean the layer is optional. - Only the **final output dimensions** are required to match exactly. - The implementation will be **manually reviewed**, so correctness of structure and logic is essential.

7.8. 0. Necessary Imports and Data Loading

```
# Cell 0.1 (DO NOT EDIT THIS CELL!)\n\nimport os\nimport torch\nimport gdown\nimport requests\nimport torch.nn as nn\nimport matplotlib.pyplot as plt\n\nfrom PIL import Image\nfrom torchvision import transforms
```

```
# Cell 0.2 (DO NOT EDIT THIS CELL!)
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device
```

```
device(type='cpu')
```

```
import torch
from torch.fx import symbolic_trace
import torch.nn as nn
import torch.nn.functional as F
from collections import Counter

# Map human-readable names to their corresponding types/functions
required_counts = {
    'Conv2d': 3,
    'ReLU': 3,
    'BatchNorm2d': 3,
    'MaxPool2d': 1,
    'Linear': 3,
    'ConvTranspose2d': 0,
    'MaxUnpool2d': 0,
    'cat': 1, # optional, set as needed
    # 'reshape': 3, # will match F.reshape, x.reshape(), nn.Flatten, nn.Unflatten,
    'LayerNorm': 1,
    'LSTM': 1,
    'Embedding': 1,
}

# Mapping from module/function type to the key in required_counts
name_map = {
    nn.Conv2d: 'Conv2d',
    nn.ReLU: 'ReLU',
    nn.BatchNorm2d: 'BatchNorm2d',
    nn.MaxPool2d: 'MaxPool2d',
    nn.Flatten: 'reshape',
    nn.Linear: 'Linear',
    nn.ConvTranspose2d: 'ConvTranspose2d',
    # nn.Unflatten: 'reshape',
    nn.MaxUnpool2d: 'MaxUnpool2d',
    # torch.flatten: 'reshape',
    torch.cat: 'cat',
    torch.add: 'add',
    # torch.reshape: 'reshape',
    F.relu: 'ReLU',
    F.batch_norm: 'BatchNorm2d', # in case batchnorm is done functionally
    F.max_pool2d: 'MaxPool2d',
    F.linear: 'Linear',
    F.conv2d: 'Conv2d',
    F.conv_transpose2d: 'ConvTranspose2d',
    F.max_unpool2d: 'MaxUnpool2d',
    nn.LSTM: 'LSTM',
    nn.Embedding: 'Embedding',
    nn.LayerNorm: 'LayerNorm',
    F.embedding: 'Embedding',
    F.linear: 'Linear',
    F.layer_norm: 'LayerNorm',
    # torch.Tensor.reshape: 'reshape',
}
```

```
def count_fx_ops(model):
    traced = symbolic_trace(model)
    op_counter = Counter()

    for node in traced.graph.nodes:
        if node.op == 'call_module':
            mod = dict(traced.named_modules())[node.target]
            name = name_map.get(type(mod))
            if name:
                op_counter[name] += 1

        elif node.op in {'call_function', 'call_method'}:
            name = name_map.get(node.target)
            if name:
                op_counter[name] += 1

    return op_counter

def check_op_counts(actual_counts, required_counts):
    issues = []
    for key, required in required_counts.items():
        actual = actual_counts.get(key, 0)
        if actual != required:
            issues.append(f"{key}: expected {required}, found {actual}")
    return issues
```

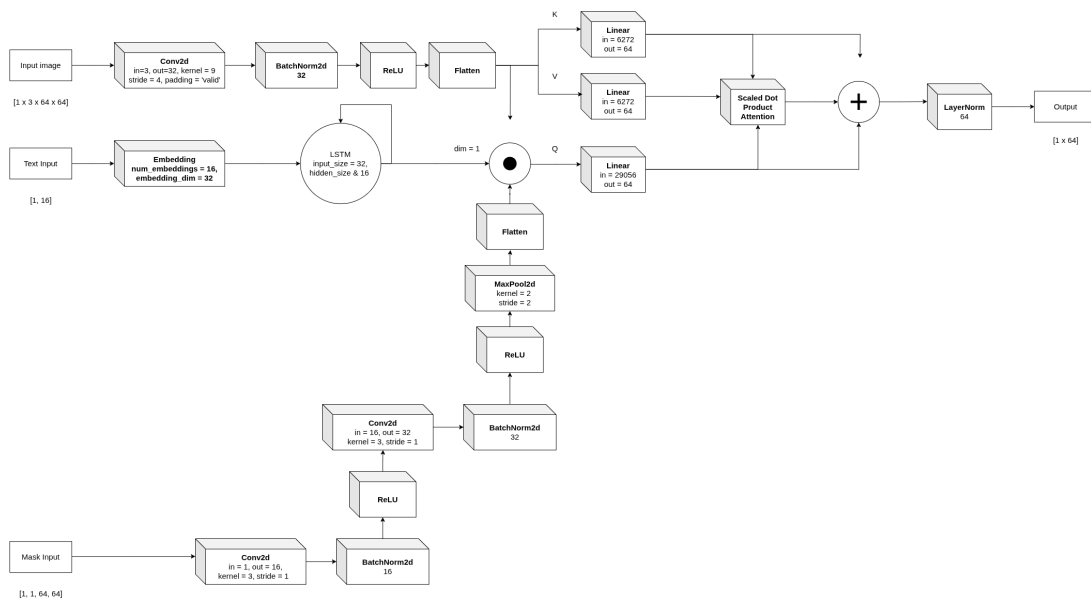
7.9. 1. Architecture

To better view the architecture diagram:

- **Right-click the image** and select “**Open image in a new tab**” to enable zoom for a clearer view.
- Alternatively, you can **download the image** using the link below:

[Download Architecture Diagram](#)

7.9.1. Diagram Preview:



```

# Cell 1.1 (EDIT THIS CELL!)

class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        # ADD YOUR CODE HERE
        # image embedding

        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=9, stride=4, padding=0)
        self.batchnorm1 = nn.BatchNorm2d(32)
        self.relu1 = nn.ReLU()

        # text embedding
        self.embedding = nn.Embedding(16, 32)
        self.lstm = nn.LSTM(32, 16, batch_first=True)

        # mask embedding
        self.conv2 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, stride=1, padding=0)
        self.batchnorm2 = nn.BatchNorm2d(16)
        self.relu2 = nn.ReLU()

        self.conv3 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=0)
        self.batchnorm3 = nn.BatchNorm2d(32)
        self.relu3 = nn.ReLU()
        self.maxpool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.flatten = nn.Flatten()

        # scaled dot product
        self.k = nn.Linear(in_features=6272, out_features=64)
        self.q = nn.Linear(in_features=29056, out_features=64)
        self.v = nn.Linear(in_features=6272, out_features=64)

        self.layernorm = nn.LayerNorm(64)

    def forward(self, image, text, mask):
        print(f"[image] Input image shape: {image.shape}")
        print(f"[text] Input text shape: {text.shape}")
        print(f"[mask] Input mask shape: {mask.shape}")

        # ADD YOUR CODE HERE

        # image embedding
        x = self.conv1(image)
        x = self.batchnorm1(x)
        x = self.relu1(x)

        x = self.flatten(x)
        print(f"[image] After embedding shape: {x.shape}")

        k = self.k(x)
        v = self.v(x)

        # text embed
        text_embedding = self.embedding(text)
        lstm_out, _ = self.lstm(text_embedding)

        print(f"[text] Text embedding shape: {lstm_out.shape}")

        # mask embed

```

```

mask_embedding = self.conv2(mask)
mask_embedding = self.batchnorm2(mask_embedding)
mask_embedding = self.relu2(mask_embedding)

mask_embedding = self.conv3(mask_embedding)
mask_embedding = self.batchnorm3(mask_embedding)
mask_embedding = self.relu3(mask_embedding)
mask_embedding = self.maxpool1(mask_embedding)

mask_embedding = self.flatten(mask_embedding)

print(f"[mask_embedding] Mask embedding shape: {mask_embedding.shape}")
# combine text embedding with mask embedding

combined_embedding = torch.cat((lstm_out.reshape((1, -1)), mask_embedding), dim=1)
print(f"[combined_embedding] Combined embedding shape: {combined_embedding.shape}")

# attention
q = self.q(combined_embedding)

print(f'[q] Q shape: {q.shape}')
print(f'[k] K shape: {k.shape}')
print(f'[v] V shape: {v.shape}')

# scaled_dot_product
sdp = torch.nn.functional.scaled_dot_product_attention(q, k, v)

sdp = sdp + q

# layernorm
x = self.layernorm(sdp)
print(f"[x] Output shape: {x.shape}")

return x

```

```

# Cell 1.2 (DO NOT EDIT THIS CELL!)

custom_model = CustomModel().to(device)
img = torch.randn(1, 3, 64, 64).to(device)
text = torch.randint(0, 16, (1, 16)).to(device)
mask = torch.randn(1, 1, 64, 64).to(device)

output = custom_model(img, text, mask)

print("Output Shape:", output.shape)

# TESTING

expected = (1, 64)
if output.shape == expected:
    print("\n Output shape is correct.\n")
else:
    print(" Output shape is incorrect. Expected: {output.shape}, Actual: {output.shape}")

actual_counts = count_fx_ops(custom_model)
issues = check_op_counts(actual_counts, required_counts)

if issues:
    print("\n Mismatch found:")
    for issue in issues:

```

```
print("-", issue)
else:
    print("\n Model seems to satisfies all module count requirements.")
```

```
[image] Input image shape: torch.Size([1, 3, 64, 64])
[text] Input text shape: torch.Size([1, 16])
[mask] Input mask shape: torch.Size([1, 1, 64, 64])
[image] After embedding shape: torch.Size([1, 6272])
[text] Text embedding shape: torch.Size([1, 16, 16])
[mask_embedding] Mask embedding shape: torch.Size([1, 28800])
[combined_embedding] Combined embedding shape: torch.Size([1, 29056])
[q] Q shape: torch.Size([1, 64])
[k] K shape: torch.Size([1, 64])
[v] V shape: torch.Size([1, 64])
[x] Output shape: torch.Size([1, 64])
Output Shape: torch.Size([1, 64])
```

Output shape is correct.

```
[image] Input image shape: Proxy(getattr_1)
[text] Input text shape: Proxy(getattr_2)
[mask] Input mask shape: Proxy(getattr_3)
[image] After embedding shape: Proxy(getattr_4)
[text] Text embedding shape: Proxy(getattr_5)
[mask_embedding] Mask embedding shape: Proxy(getattr_6)
[combined_embedding] Combined embedding shape: Proxy(getattr_7)
[q] Q shape: Proxy(getattr_8)
[k] K shape: Proxy(getattr_9)
[v] V shape: Proxy(getattr_10)
[x] Output shape: Proxy(getattr_11)
```

Model seems to satisfies all module count requirements.

Chapter 8

Entry Exam 2025-06-17, Variant 1

8.1. Student Information

- **Name:** <Enter your full name>
 - **Neptun ID:** <Enter your Neptun ID>
 - **Date:** 17/06/2025

 - **Time Slot:** 09:00 - 09:30 (30 minutes)
-

8.2. Overview

This coding task must be completed to qualify for the second part of the final exam (1-hour written session). Final course grades will be assigned based on overall performance during the active semester, provided both components are passed.

8.3. Task Requirements

- Implement a **neural network architecture**, including a **functioning forward pass**.
 - The code must run without errors and meet the minimum implementation criteria.
-

8.4. Rules and Resources

Allowed	Not Allowed
Internet access	Communication with others (e.g., Teams, WhatsApp, Messenger)
AI tools (e.g., ChatGPT, Copilot)	Collaboration or assistance from peers
Practice notebooks	Use of group chats, forums, or messaging apps

Any violation will result in **immediate disqualification** and a **failing grade**.

8.5. Submission Guidelines

- Submit your solution as a **.ipynb file** on **Canvas**
 - Late submissions or incomplete solutions will not be accepted
 - Your code must be fully executable and meet the minimum task requirements
-

8.6. Retake Policy

- If this part is failed, **one retake opportunity** will be provided
 - Failing the retake results in a **fail for the course**
-

Good luck!

8.7. Task Description

Your task is to implement a **custom neural network architecture**, including a fully functional `forward()` method.

You will be provided with a **diagram of the model architecture**. Some components will contain a **?** **symbol**, indicating that you must determine and specify the appropriate hyperparameters or configurations based on the overall architecture and problem.

Please note the following: - **All layers shown in the diagram must be included** in your implementation. - If the shapes **do not match** check the set parameters, **interpolation and resize** do not considered a valid mitigation of the issue. - If you use **torch.nn.functional** it counts as a **layer or block**. - In some cases, **connections may not be fully drawn**, but this does not mean the layer is optional. - Only the **final output dimensions** are required to match exactly. - The implementation will be **manually reviewed**, so correctness of structure and logic is essential.

8.8. 0. Necessary Imports and Data Loading

```
# Cell 0.1 (DO NOT EDIT THIS CELL!)

import os
import torch
import gdown
import requests
import torch.nn as nn
import matplotlib.pyplot as plt

from PIL import Image
from torchvision import transforms
```

```
# Cell 0.2 (DO NOT EDIT THIS CELL!)
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device
```

```
device(type='cuda')
```

```
# Cell 0.3 -> INPUT (DO NOT EDIT THIS CELL!)
```

```
url = 'https://drive.google.com/uc?id=1NiPgDz4eDsJxU5WKLWUFIJEEp-UUCmu-'
output = 'input_image.png'
```

```
try:
```

```
    gdown.download(url, output, quiet=False)
    input_image = Image.open(output).convert('RGB')
```

```
    transform = transforms.Compose([
        transforms.Resize((128, 128)),
        transforms.ToTensor(),
    ])
```

```
    input_tensor = transform(input_image).unsqueeze(0).to(device)
```

```
except:
```

```
    input_tensor = torch.randn(1, 3, 128, 128)
    input_image = transforms.ToPILImage()(input_tensor.squeeze(0).to(device))
```

```
Downloading...
```

```
From: https://drive.google.com/uc?id=1NiPgDz4eDsJxU5WKLWUFIJEEp-UUCmu-
```

```
To: /content/input_image.png
```

```
100%|#####| 419k/419k [00:00<00:00, 120MB/s]
```

```
# Cell 0.4 -> INPUT (DO NOT EDIT THIS CELL!)
```

```
vocab_size = 16
input_tokens = torch.randint(0, vocab_size, (1, 16)).to(device)
print(input_tokens.shape)
```

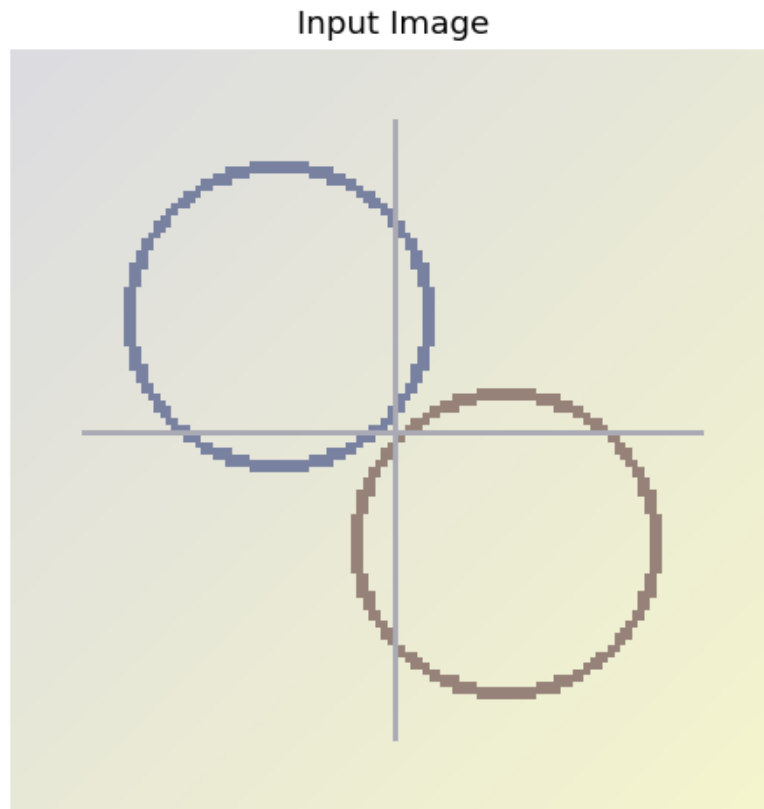
```
torch.Size([1, 16])
```

```
# Cell 0.5 (DO NOT EDIT THIS CELL!)
```

```
print("Tensor shape:", input_tensor.shape)
```

```
plt.imshow(input_image)
plt.axis('off')
plt.title("Input Image")
plt.show()
```

```
Tensor shape: torch.Size([1, 3, 128, 128])
```



```
# Cell 0.3 (DO NOT EDIT THIS CELL!)

import torch
import io
import torch.nn as nn
import torch.nn.functional as F
from torch.fx import symbolic_trace
from contextlib import redirect_stdout
from collections import Counter

required_counts = {
    'Conv2d': 6,
    'ReLU': 5,
    'BatchNorm2d': 2,
    'AdaptiveAvgPool2d': 1,
    'Linear': 4,
    'MultiHeadAttention': 1,
    'Embedding': 1,
    'LSTM': 1,
    'cat': 1,
    'reshape': 2,
}

name_map = {
    nn.Conv1d: 'Conv1d',
    nn.Conv2d: 'Conv2d',
    nn.Conv3d: 'Conv3d',
    nn.ConvTranspose1d: 'ConvTranspose1d',
    nn.ConvTranspose2d: 'ConvTranspose2d',
```

```

nn.ConvTranspose3d: 'ConvTranspose3d',
nn.BatchNorm1d: 'BatchNorm1d',
nn.BatchNorm2d: 'BatchNorm2d',
nn.BatchNorm3d: 'BatchNorm3d',
nn.ReLU: 'ReLU',
nn.ReLU6: 'ReLU6',
nn.Sigmoid: 'Sigmoid',
nn.Tanh: 'Tanh',
nn.ELU: 'ELU',
nn.LeakyReLU: 'LeakyReLU',
nn.GELU: 'GELU',
nn.SELU: 'SELU',
nn.PReLU: 'PReLU',
nn.Softmax: 'Softmax',
nn.LogSoftmax: 'LogSoftmax',
nn.Softplus: 'Softplus',
nn.Softsign: 'Softsign',
nn.Dropout: 'Dropout',
nn.Dropout2d: 'Dropout2d',
nn.Dropout3d: 'Dropout3d',
nn.AlphaDropout: 'AlphaDropout',
nn.MaxPool1d: 'MaxPool1d',
nn.MaxPool2d: 'MaxPool2d',
nn.MaxPool3d: 'MaxPool3d',
nn.AvgPool1d: 'AvgPool1d',
nn.AvgPool2d: 'AvgPool2d',
nn.AvgPool3d: 'AvgPool3d',
nn.AdaptiveAvgPool1d: 'AdaptiveAvgPool1d',
nn.AdaptiveAvgPool2d: 'AdaptiveAvgPool2d',
nn.AdaptiveAvgPool3d: 'AdaptiveAvgPool3d',
nn.AdaptiveMaxPool1d: 'AdaptiveMaxPool1d',
nn.AdaptiveMaxPool2d: 'AdaptiveMaxPool2d',
nn.AdaptiveMaxPool3d: 'AdaptiveMaxPool3d',
nn.Linear: 'Linear',
nn.Bilinear: 'Bilinear',
nn.Flatten: 'reshape',
nn.Unflatten: 'reshape',
nn.Embedding: 'Embedding',
nn.EmbeddingBag: 'EmbeddingBag',
nn.LSTM: 'LSTM',
nn.GRU: 'GRU',
nn.RNN: 'RNN',
nn.Upsample: 'Upsample',
nn.MultiheadAttention: 'MultiHeadAttention',
nn.LayerNorm: 'LayerNorm',
nn.GroupNorm: 'GroupNorm',
nn.InstanceNorm1d: 'InstanceNorm1d',
nn.InstanceNorm2d: 'InstanceNorm2d',
nn.InstanceNorm3d: 'InstanceNorm3d',

F.relu: 'ReLU',
F.relu6: 'ReLU6',
F.sigmoid: 'Sigmoid',
F.tanh: 'Tanh',
F.elu: 'ELU',
F.leaky_relu: 'LeakyReLU',
F.gelu: 'GELU',
F.selu: 'SELU',
F.prelu: 'PReLU',
F.softmax: 'Softmax',

```

```

F.log_softmax: 'LogSoftmax',
F.softplus: 'Softplus',
F.softsign: 'Softsign',
F.dropout: 'Dropout',
F.alpha_dropout: 'AlphaDropout',
F.max_pool1d: 'MaxPool1d',
F.max_pool2d: 'MaxPool2d',
F.max_pool3d: 'MaxPool3d',
F.avg_pool1d: 'AvgPool1d',
F.avg_pool2d: 'AvgPool2d',
F.avg_pool3d: 'AvgPool3d',
F.adaptive_avg_pool1d: 'AdaptiveAvgPool1d',
F.adaptive_avg_pool2d: 'AdaptiveAvgPool2d',
F.adaptive_avg_pool3d: 'AdaptiveAvgPool3d',
F.adaptive_max_pool1d: 'AdaptiveMaxPool1d',
F.adaptive_max_pool2d: 'AdaptiveMaxPool2d',
F.adaptive_max_pool3d: 'AdaptiveMaxPool3d',
F.batch_norm: 'BatchNorm',
F.instance_norm: 'InstanceNorm',
F.layer_norm: 'LayerNorm',
F.group_norm: 'GroupNorm',
F.linear: 'Linear',
F.bilinear: 'Bilinear',
F.conv1d: 'Conv1d',
F.conv2d: 'Conv2d',
F.conv3d: 'Conv3d',
F.conv_transpose1d: 'ConvTranspose1d',
F.conv_transpose2d: 'ConvTranspose2d',
F.conv_transpose3d: 'ConvTranspose3d',
F.embedding: 'Embedding',
F.embedding_bag: 'EmbeddingBag',
F.dropout2d: 'Dropout2d',
F.dropout3d: 'Dropout3d',
torch.flatten: 'reshape',
torch.cat: 'cat',
torch.stack: 'stack',
torch.add: 'add',
torch.sub: 'sub',
torch.mul: 'mul',
torch.div: 'div',
torch.mean: 'mean',
torch.sum: 'sum',
torch.reshape: 'reshape',
torch.transpose: 'reshape',
torch.permute: 'reshape',
torch.nn.functional.normalize: 'normalize',

'view': 'reshape',
'reshape': 'reshape',
'flatten': 'reshape',
'permute': 'reshape',
'transpose': 'reshape'
}

def count_fx_ops(model):
    f = io.StringIO()
    with redirect_stdout(f):
        traced = symbolic_trace(model)
    op_counter = Counter()

```

```

for node in traced.graph.nodes:
    if node.op == 'call_module':
        mod = dict(traced.named_modules())[node.target]
        name = name_map.get(type(mod))
        if name:
            op_counter[name] += 1

    elif node.op in {'call_function', 'call_method'}:
        name = name_map.get(node.target)
        if name:
            op_counter[name] += 1

return op_counter

def check_op_counts(actual_counts, required_counts):
    issues = []
    for key, required in required_counts.items():
        actual = actual_counts.get(key, 0)
        if actual != required:
            issues.append(f"{key}: expected {required}, found {actual}")
    for key in actual_counts:
        if key not in required_counts:
            issues.append(f"{key}: unexpected layer found (count = {actual_counts[key]})")
    return issues

```

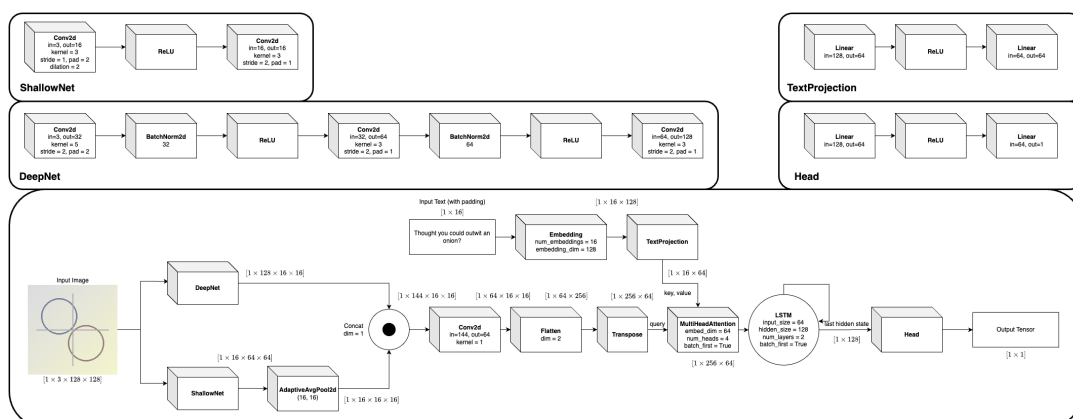
8.9. 1. Architecture

To better view the architecture diagram:

- **Right-click the image** and select “**Open image in a new tab**” to enable zoom for a clearer view.
- Alternatively, you can **download the image** using the link below:

[Download Architecture Diagram](#)

8.9.1. Diagram Preview:



Cell 1.1 (EDIT THIS CELL!)

```

class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()

```

```

self.vision_deep = nn.Sequential(
    nn.Conv2d(3, 32, kernel_size=5, stride=2, padding=2),
    nn.BatchNorm2d(32),
    nn.ReLU(),
    nn.Conv2d(32, 64, kernel_size=3, stride=2, padding=1),
    nn.BatchNorm2d(64),
    nn.ReLU(),
    nn.Conv2d(64, 128, kernel_size=3, stride=2, padding=1),
)

self.vision_shallow = nn.Sequential(
    nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=2, dilation=2),
    nn.ReLU(),
    nn.Conv2d(16, 16, kernel_size=3, stride=2, padding=1),
)

self.fusion_conv = nn.Conv2d(144, 64, kernel_size=1)
self.embedding = nn.Embedding(num_embeddings=16, embedding_dim=128)
self.text_proj = nn.Sequential(
    nn.Linear(128, 64),
    nn.ReLU(),
    nn.Linear(64, 64)
)

self.cross_attention = nn.MultiheadAttention(embed_dim=64, num_heads=4, batch_first=True)
self.lstm = nn.LSTM(input_size=64, hidden_size=128, num_layers=2, batch_first=True)
self.adaptive_pool = nn.AdaptiveAvgPool2d((16, 16))

self.output_head = nn.Sequential(
    nn.Linear(128, 64),
    nn.ReLU(),
    nn.Linear(64, 1)
)

def forward(self, image, text):
    print(f"[image] Input shape: {image.shape}")
    print(f"[text] Input shape: {text.shape}")

    x_deep = self.vision_deep(image)
    print(f"[x_deep] After deep CNN branch: {x_deep.shape}")
    x_shallow = self.vision_shallow(image)
    print(f"[x_shallow] After shallow CNN branch: {x_shallow.shape}")
    x_shallow = self.adaptive_pool(x_shallow)
    print(f"[x_shallow] After adaptive pooling: {x_shallow.shape}")
    x = torch.cat([x_deep, x_shallow], dim=1)
    print(f"[x] After concatenating branches: {x.shape}")
    x = self.fusion_conv(x)
    print(f"[x] After 1x1 fusion convolution: {x.shape}")
    x = torch.flatten(x, start_dim=2, end_dim=3)
    x = torch.transpose(x, 1, 2)
    print(f"[x] After flattening to sequence: {x.shape}")
    y = self.embedding(text)
    print(f"[y] After embedding: {y.shape}")
    y = self.text_proj(y)
    print(f"[y] After projection: {y.shape}")
    attn_out, _ = self.cross_attention(query=x, key=y, value=y)
    print(f"[attn_out] After cross-attention: {attn_out.shape}")
    lstm_out, _ = self.lstm(attn_out)
    print(f"[lstm_out] After LSTM: {lstm_out.shape}")

```

```

print(f"[lstm_out] {lstm_out[:, -1, :].shape}")
out = self.output_head(lstm_out[:, -1, :])
print(f"[out] Final output shape: {out.shape}")
return out

```

Cell 1.2 (DO NOT EDIT THIS CELL!)

```
custom_model = CustomModel().to(device)
```

```
output = custom_model(input_tensor, input_tokens)
```

```
try:
```

```
    actual_counts = count_fx_ops(custom_model)
```

```
    issues = check_op_counts(actual_counts, required_counts)
```

```
    if output.shape != (1, 1):
```

```
        raise AssertionError("Output shape is incorrect.")
```

```
    if issues:
```

```
        raise AssertionError("Operation count mismatch.")
```

```
    print("\n Congratulations! Your implementation is correct. You passed the first part of the exam! ")
```

```
except AssertionError as e:
```

```
    print(f"\n Error: {e}")
```

```
    if output.shape != (1, 1):
```

```
        print(f" - Expected output shape (1, 1), but got {output.shape}")
```

```
    if issues:
```

```
        print(" Mismatch in operation counts:")
```

```
        for issue in issues:
```

```
            print(" -", issue)
```

```
[image] Input shape: torch.Size([1, 3, 128, 128])
```

```
[text] Input shape: torch.Size([1, 16])
```

```
[x_deep] After deep CNN branch: torch.Size([1, 128, 16, 16])
```

```
[x_shallow] After shallow CNN branch: torch.Size([1, 16, 64, 64])
```

```
[x_shallow] After adaptive pooling: torch.Size([1, 16, 16, 16])
```

```
[x] After concatenating branches: torch.Size([1, 144, 16, 16])
```

```
[x] After 1x1 fusion convolution: torch.Size([1, 64, 16, 16])
```

```
[x] After flattening to sequence: torch.Size([1, 256, 64])
```

```
[y] After embedding: torch.Size([1, 16, 128])
```

```
[y] After projection: torch.Size([1, 16, 64])
```

```
[attn_out] After cross-attention: torch.Size([1, 256, 64])
```

```
[lstm_out] After LSTM: torch.Size([1, 256, 128])
```

```
[lstm_out] torch.Size([1, 128])
```

```
[out] Final output shape: torch.Size([1, 1])
```

Congratulations! Your implementation is correct. You passed the first part of the exam!

Chapter 9

Entry Exam 2025-06-17, Variant 2

9.1. Student Information

- **Name:** <Enter your full name>
 - **Neptun ID:** <Enter your Neptun ID>
 - **Date:** 17/06/2025

 - **Time Slot:** 09:00 - 09:30 (30 minutes)
-

9.2. Overview

This coding task must be completed to qualify for the second part of the final exam (1-hour written session). Final course grades will be assigned based on overall performance during the active semester, provided both components are passed.

9.3. Task Requirements

- Implement a **neural network architecture**, including a **functioning forward pass**.
 - The code must run without errors and meet the minimum implementation criteria.
-

9.4. Rules and Resources

Allowed	Not Allowed
Internet access	Communication with others (e.g., Teams, WhatsApp, Messenger)
AI tools (e.g., ChatGPT, Copilot)	Collaboration or assistance from peers
Practice notebooks	Use of group chats, forums, or messaging apps

Any violation will result in **immediate disqualification** and a **failing grade**.

9.5. Submission Guidelines

- Submit your solution as a **.ipynb file** on **Canvas**
 - Late submissions or incomplete solutions will not be accepted
 - Your code must be fully executable and meet the minimum task requirements
-

9.6. Retake Policy

- If this part is failed, **one retake opportunity** will be provided
 - Failing the retake results in a **fail for the course**
-

Good luck!

9.7. Task Description

Your task is to implement a **custom neural network architecture**, including a fully functional `forward()` method.

You will be provided with a **diagram of the model architecture**. Some components will contain a **?** **symbol**, indicating that you must determine and specify the appropriate hyperparameters or configurations based on the overall architecture and problem.

Please note the following:

- **All layers shown in the diagram must be included** in your implementation.
- If the shapes **do not match** check the set parameters, **interpolation and resize** do not considered a valid mitigation of the issue.
- If you use **torch.nn.functional** it counts as a **layer or block**.
- In some cases, **connections may not be fully drawn**, but this does not mean the layer is optional.
- Only the **final output dimensions** are required to match exactly.
- The implementation will be **manually reviewed**, so correctness of structure and logic is essential.

9.8. 0. Necessary Imports and Data Loading

```
# Cell 0.1 (DO NOT EDIT THIS CELL!)

import os
import torch
import gdown
import requests
import torch.nn as nn
import matplotlib.pyplot as plt

from PIL import Image
from torchvision import transforms

import torch
```

```
import torch.nn as nn
import torch.nn.functional as F
```

```
# Cell 0.2 (DO NOT EDIT THIS CELL!)
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device
```

```
device(type='cuda')
```

```
# Cell 0.3 (DO NOT EDIT THIS CELL!)
```

```
import torch
import io
import torch.nn as nn
import torch.nn.functional as F
from torch.fx import symbolic_trace
from contextlib import redirect_stdout
from collections import Counter
```

```
required_counts = {
    'Conv2d': 6,
    'ReLU': 1,
    'LeakyReLU': 3,
    'Dropout': 1,
    'MaxPool2d': 2,
    'Linear': 5,
    'ConvTranspose2d': 1,
    'Upsample': 1,
    'cat': 2,
    'add': 1,
    'reshape': 3,
}
```

```
name_map = {
    nn.Conv1d: 'Conv1d',
    nn.Conv2d: 'Conv2d',
    nn.Conv3d: 'Conv3d',
    nn.ConvTranspose1d: 'ConvTranspose1d',
    nn.ConvTranspose2d: 'ConvTranspose2d',
    nn.ConvTranspose3d: 'ConvTranspose3d',
    nn.BatchNorm1d: 'BatchNorm1d',
    nn.BatchNorm2d: 'BatchNorm2d',
    nn.BatchNorm3d: 'BatchNorm3d',
    nn.ReLU: 'ReLU',
    nn.ReLU6: 'ReLU6',
    nn.Sigmoid: 'Sigmoid',
    nn.Tanh: 'Tanh',
    nn.ELU: 'ELU',
    nn.LeakyReLU: 'LeakyReLU',
    nn.GELU: 'GELU',
    nn.SELU: 'SELU',
    nn.PReLU: 'PReLU',
    nn.Softmax: 'Softmax',
    nn.LogSoftmax: 'LogSoftmax',
    nn.Softplus: 'Softplus',
    nn.Softsign: 'Softsign',
    nn.Dropout: 'Dropout',
    nn.Dropout2d: 'Dropout2d',
    nn.Dropout3d: 'Dropout3d',
```

```

nn.AlphaDropout: 'AlphaDropout',
nn.MaxPool1d: 'MaxPool1d',
nn.MaxPool2d: 'MaxPool2d',
nn.MaxPool3d: 'MaxPool3d',
nn.AvgPool1d: 'AvgPool1d',
nn.AvgPool2d: 'AvgPool2d',
nn.AvgPool3d: 'AvgPool3d',
nn.AdaptiveAvgPool1d: 'AdaptiveAvgPool1d',
nn.AdaptiveAvgPool2d: 'AdaptiveAvgPool2d',
nn.AdaptiveAvgPool3d: 'AdaptiveAvgPool3d',
nn.AdaptiveMaxPool1d: 'AdaptiveMaxPool1d',
nn.AdaptiveMaxPool2d: 'AdaptiveMaxPool2d',
nn.AdaptiveMaxPool3d: 'AdaptiveMaxPool3d',
nn.Linear: 'Linear',
nn.Bilinear: 'Bilinear',
nn.Flatten: 'reshape',
nn.Unflatten: 'reshape',
nn.Embedding: 'Embedding',
nn.EmbeddingBag: 'EmbeddingBag',
nn.LSTM: 'LSTM',
nn.GRU: 'GRU',
nn.RNN: 'RNN',
nn.Upsample: 'Upsample',
nn.MultiheadAttention: 'MultiHeadAttention',
nn.LayerNorm: 'LayerNorm',
nn.GroupNorm: 'GroupNorm',
nn.InstanceNorm1d: 'InstanceNorm1d',
nn.InstanceNorm2d: 'InstanceNorm2d',
nn.InstanceNorm3d: 'InstanceNorm3d',

F.relu: 'ReLU',
F.relu6: 'ReLU6',
F.sigmoid: 'Sigmoid',
F.tanh: 'Tanh',
F.elu: 'ELU',
F.leaky_relu: 'LeakyReLU',
F.gelu: 'GELU',
F.selu: 'SELU',
F.prelu: 'PReLU',
F.softmax: 'Softmax',
F.log_softmax: 'LogSoftmax',
F.softplus: 'Softplus',
F.softsign: 'Softsign',
F.dropout: 'Dropout',
F.alpha_dropout: 'AlphaDropout',
F.max_pool1d: 'MaxPool1d',
F.max_pool2d: 'MaxPool2d',
F.max_pool3d: 'MaxPool3d',
F.avg_pool1d: 'AvgPool1d',
F.avg_pool2d: 'AvgPool2d',
F.avg_pool3d: 'AvgPool3d',
F.adaptive_avg_pool1d: 'AdaptiveAvgPool1d',
F.adaptive_avg_pool2d: 'AdaptiveAvgPool2d',
F.adaptive_avg_pool3d: 'AdaptiveAvgPool3d',
F.adaptive_max_pool1d: 'AdaptiveMaxPool1d',
F.adaptive_max_pool2d: 'AdaptiveMaxPool2d',
F.adaptive_max_pool3d: 'AdaptiveMaxPool3d',
F.batch_norm: 'BatchNorm',
F.instance_norm: 'InstanceNorm',
F.layer_norm: 'LayerNorm',

```

```

F.group_norm: 'GroupNorm',
F.linear: 'Linear',
F.bilinear: 'Bilinear',
F.conv1d: 'Conv1d',
F.conv2d: 'Conv2d',
F.conv3d: 'Conv3d',
F.conv_transpose1d: 'ConvTranspose1d',
F.conv_transpose2d: 'ConvTranspose2d',
F.conv_transpose3d: 'ConvTranspose3d',
F.embedding: 'Embedding',
F.embedding_bag: 'EmbeddingBag',
F.dropout2d: 'Dropout2d',
F.dropout3d: 'Dropout3d',
torch.flatten: 'reshape',
torch.cat: 'cat',
torch.stack: 'stack',
torch.add: 'add',
torch.sub: 'sub',
torch.mul: 'mul',
torch.div: 'div',
torch.mean: 'mean',
torch.sum: 'sum',
torch.reshape: 'reshape',
torch.transpose: 'reshape',
torch.permute: 'reshape',
torch.nn.functional.normalize: 'normalize',

'view': 'reshape',
'reshape': 'reshape',
'flatten': 'reshape',
'permute': 'reshape',
'transpose': 'reshape'
}

def count_fx_ops(model):
    f = io.StringIO()
    with redirect_stdout(f):
        traced = symbolic_trace(model)
        op_counter = Counter()

    for node in traced.graph.nodes:
        if node.op == 'call_module':
            mod = dict(traced.named_modules())[node.target]
            name = name_map.get(type(mod))
            if name:
                op_counter[name] += 1

        elif node.op in {'call_function', 'call_method'}:
            name = name_map.get(node.target)
            if name:
                op_counter[name] += 1

    return op_counter

def check_op_counts(actual_counts, required_counts):
    issues = []
    for key, required in required_counts.items():
        actual = actual_counts.get(key, 0)
        if actual != required:
            issues.append(f"{key}: expected {required}, found {actual}")

```

```

for key in actual_counts:
    if key not in required_counts:
        issues.append(f"{key}: unexpected layer found (count = {actual_counts[key]})")
return issues

```

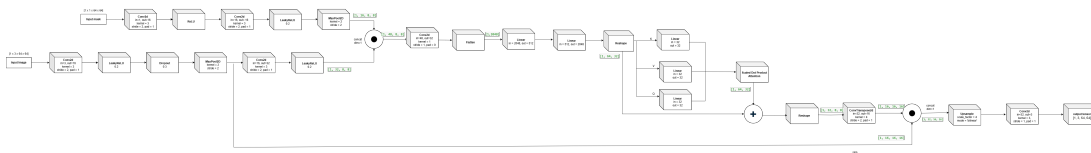
9.9. 1. Architecture

To better view the architecture diagram:

- **Right-click the image** and select “**Open image in a new tab**” to enable zoom for a clearer view.
- Alternatively, you can **download the image** using the link below:

[Download Architecture Diagram](#)

9.9.1. Diagram Preview:



Cell 1.1 (EDIT THIS CELL!)

```

class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1_img = nn.Conv2d(3, 16, kernel_size=3, stride=2, padding=1)
        self.act1_img = nn.LeakyReLU(0.2)
        self.drop1_img = nn.Dropout(0.3)
        self.pool_img = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2_img = nn.Conv2d(16, 32, kernel_size=3, stride=2, padding=1)
        self.act2_img = nn.LeakyReLU(0.2)

        self.conv1_mask = nn.Conv2d(1, 16, kernel_size=3, stride=2, padding=1)
        self.act1_mask = nn.ReLU()
        self.conv2_mask = nn.Conv2d(16, 16, kernel_size=3, stride=2, padding=1)
        self.act2_mask = nn.LeakyReLU(0.2)
        self.pool_mask = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv1x1 = nn.Conv2d(48, 32, kernel_size=1, stride=1, padding=0)
        self.linear1 = nn.Linear(32 * 8 * 8, 512)
        self.linear2 = nn.Linear(512, 32 * 8 * 8)

        self.query_linear = nn.Linear(32, 32)
        self.key_linear = nn.Linear(32, 32)
        self.value_linear = nn.Linear(32, 32)

        self.deconv1 = nn.ConvTranspose2d(32, 16, kernel_size=4, stride=2, padding=1)
        self.upsample = nn.Upsample(scale_factor=4, mode='bilinear')
        self.final_conv = nn.Conv2d(32, 3, kernel_size=3, stride=1, padding=1)

    def forward(self, image, mask):
        print(f"[image] Input image shape: {image.shape}")
        print(f"[mask] Input mask shape: {input_mask.shape}")
        x_img = self.conv1_img(image)

```

```

x_img = self.act1_img(x_img)
x_img = self.drop1_img(x_img)
print(f"[x_img] After conv1+act+drop: {x_img.shape}")

x_img = self.pool_img(x_img)
print(f"[x_img] After pool: {x_img.shape}")
skip2 = x_img

x_img = self.conv2_img(x_img)
x_img = self.act2_img(x_img)
print(f"[x_img] After conv2+act: {x_img.shape}")

x_mask = self.conv1_mask(mask)
x_mask = self.act1_mask(x_mask)
x_mask = self.conv2_mask(x_mask)
x_mask = self.act2_mask(x_mask)
x_mask = self.pool_mask(x_mask)
print(f"[x_mask] After mask branch: {x_mask.shape}")

x = torch.cat((x_img, x_mask), dim=1)
print(f"[x] After concat: {x.shape}")

x = self.conv_1x1(x)
print(f"[x] After 1x1 conv: {x.shape}")

x = x.view(x.size(0), -1)
print(f"[x] After flatten: {x.shape}")

x = self.linear1(x)
x = self.linear2(x)
print(f"[x] After linear layers: {x.shape}")

x = x.view(x.size(0), 64, 32)
print(f"[x] After reshape to sequence: {x.shape}")

query = self.query_linear(x)
key = self.key_linear(x)
value = self.value_linear(x)
print(f"[query/key/value] Shape: {query.shape}")

attn_out = F.scaled_dot_product_attention(query, key, value)
print(f"[attn_out] After attention: {attn_out.shape}")

x = torch.add(x, attn_out)
print(f"[x] After residual connection: {x.shape}")

x = x.view(x.size(0), 32, 8, 8)
print(f"[x] After reshape to feature map: {x.shape}")

x = self.deconv1(x)
print(f"[x] After deconv: {x.shape}")

x = torch.cat((x, skip2), dim=1)
print(f"[x] After concat with skip2: {x.shape}")

x = self.upsample(x)
print(f"[x] After upsample: {x.shape}")

x = self.final_conv(x)
print(f"[x] Final output: {x.shape}")

```

```
return x
```

```
# Cell 1.2 (DO NOT EDIT THIS CELL!)
```

```
input_image = torch.randn(1, 3, 64, 64).to(device)
input_mask = torch.randn(1, 1, 64, 64).to(device)

custom_model = CustomModel().to(device)

output = custom_model(input_image , input_mask)

try:
    actual_counts = count_fx_ops(custom_model)
    issues = check_op_counts(actual_counts, required_counts)

    if output.shape != (1, 3, 64, 64):
        raise AssertionError("Output shape is incorrect.")
    if issues:
        raise AssertionError("Operation count mismatch.")

    print("\n Congratulations! Your implementation is correct. You passed the first part of the exam! ")

except AssertionError as e:
    print(f"\n Error: {e}")
    if output.shape != (1, 3, 64, 64):
        print(f" - Expected output shape (1, 3, 64, 64), but got {output.shape}")
    if issues:
        print(" Mismatch in operation counts:")
        for issue in issues:
            print(" -", issue)
```

```
[image] Input image shape: torch.Size([1, 3, 64, 64])
[mask] Input mask shape: torch.Size([1, 1, 64, 64])
[x_img] After conv1+act+drop: torch.Size([1, 16, 32, 32])
[x_img] After pool: torch.Size([1, 16, 16, 16])
[x_img] After conv2+act: torch.Size([1, 32, 8, 8])
[x_mask] After mask branch: torch.Size([1, 16, 8, 8])
[x] After concat: torch.Size([1, 48, 8, 8])
[x] After 1x1 conv: torch.Size([1, 32, 8, 8])
[x] After flatten: torch.Size([1, 2048])
[x] After linear layers: torch.Size([1, 2048])
[x] After reshape to sequence: torch.Size([1, 64, 32])
[query/key/value] Shape: torch.Size([1, 64, 32])
[attn_out] After attention: torch.Size([1, 64, 32])
[x] After residual connection: torch.Size([1, 64, 32])
[x] After reshape to feature map: torch.Size([1, 32, 8, 8])
[x] After deconv: torch.Size([1, 16, 16, 16])
[x] After concat with skip2: torch.Size([1, 32, 16, 16])
[x] After upsample: torch.Size([1, 32, 64, 64])
[x] Final output: torch.Size([1, 3, 64, 64])
```

Congratulations! Your implementation is correct. You passed the first part of the exam!

Chapter 10

Entry Exam 2025-06-30, Variant 1

10.1. Student Information

- **Name:** *<Enter your full name>*
 - **Neptun ID:** *<Enter your Neptun ID>*
 - **Date:** 30/06/2025

 - **Time Slot:** 09:00 - 09:30 (30 minutes)
-

10.2. Overview

This coding task must be completed to qualify for the second part of the final exam (1-hour written session). Final course grades will be assigned based on overall performance during the active semester, provided both components are passed.

10.3. Task Requirements

- Implement a **neural network architecture**, including a **functioning forward pass**.
 - The code must run without errors and meet the minimum implementation criteria.
-

10.4. Rules and Resources

Allowed	Not Allowed
Internet access	Communication with others (e.g., Teams, WhatsApp, Messenger)
AI tools (e.g., ChatGPT, Copilot)	Collaboration or assistance from peers
Practice notebooks	Use of group chats, forums, or messaging apps

Any violation will result in **immediate disqualification** and a **failing grade**.

10.5. Submission Guidelines

- Submit your solution as a **.ipynb file** on **Canvas**
 - Late submissions or incomplete solutions will not be accepted
 - Your code must be fully executable and meet the minimum task requirements
-

10.6. Retake Policy

- If this part is failed, **one retake opportunity** will be provided
 - Failing the retake results in a **fail for the course**
-

Good luck!

10.7. Task Description

Your task is to implement a **custom neural network architecture**, including a fully functional `forward()` method.

You will be provided with a **diagram of the model architecture**. Some components will contain a **?** **symbol**, indicating that you must determine and specify the appropriate hyperparameters or configurations based on the overall architecture and problem.

Please note the following:

- **All layers shown in the diagram must be included** in your implementation.
- If the shapes **do not match** check the set parameters, **interpolation and resize** do not considered a valid mitigation of the issue.
- If you use **torch.nn.functional** it counts as a **layer or block**.
- In some cases, **connections may not be fully drawn**, but this does not mean the layer is optional.
- Only the **final output dimensions** are required to match exactly.
- The implementation will be **manually reviewed**, so correctness of structure and logic is essential.

10.8. 0. Necessary Imports and Data Loading

```
# Cell 0.1 (DO NOT EDIT THIS CELL!)

import os
import torch
import gdown
import requests
import torch.nn as nn
import matplotlib.pyplot as plt

from PIL import Image
from torchvision import transforms
```

```
# Cell 0.2 (DO NOT EDIT THIS CELL!)
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device
```

```
device(type='cpu')
```

```
# Cell 0.3 (DO NOT EDIT THIS CELL!)
```

```
import torch
import io
import torch.nn as nn
import torch.nn.functional as F
from torch.fx import symbolic_trace
from contextlib import redirect_stdout
from collections import Counter

required_counts = {
    'Conv2d': 3,
    'GELU': 1,
    'BatchNorm2d': 2,
    'Linear': 2,
    'Embedding': 1,
    'LSTM': 1,
    'reshape': 8,
    'LeakyReLU': 1,
    'cat': 1,
    'reshape': 3,
    'MaxPool2d': 1,
    'ReLU': 1,
    'ConvTranspose2d': 1
}

name_map = {
    nn.Conv1d: 'Conv1d',
    nn.Conv2d: 'Conv2d',
    nn.Conv3d: 'Conv3d',
    nn.ConvTranspose1d: 'ConvTranspose1d',
    nn.ConvTranspose2d: 'ConvTranspose2d',
    nn.ConvTranspose3d: 'ConvTranspose3d',
    nn.BatchNorm1d: 'BatchNorm1d',
    nn.BatchNorm2d: 'BatchNorm2d',
    nn.BatchNorm3d: 'BatchNorm3d',
    nn.ReLU: 'ReLU',
    nn.ReLU6: 'ReLU6',
    nn.Sigmoid: 'Sigmoid',
    nn.Tanh: 'Tanh',
    nn.ELU: 'ELU',
    nn.LeakyReLU: 'LeakyReLU',
    nn.GELU: 'GELU',
    nn.SELU: 'SELU',
    nn.PReLU: 'PReLU',
    nn.Softmax: 'Softmax',
    nn.LogSoftmax: 'LogSoftmax',
    nn.Softplus: 'Softplus',
    nn.Softsign: 'Softsign',
    nn.Dropout: 'Dropout',
    nn.Dropout2d: 'Dropout2d',
    nn.Dropout3d: 'Dropout3d',
    nn.AlphaDropout: 'AlphaDropout',
```

```

nn.MaxPool1d: 'MaxPool1d',
nn.MaxPool2d: 'MaxPool2d',
nn.MaxPool3d: 'MaxPool3d',
nn.AvgPool1d: 'AvgPool1d',
nn.AvgPool2d: 'AvgPool2d',
nn.AvgPool3d: 'AvgPool3d',
nn.AdaptiveAvgPool1d: 'AdaptiveAvgPool1d',
nn.AdaptiveAvgPool2d: 'AdaptiveAvgPool2d',
nn.AdaptiveAvgPool3d: 'AdaptiveAvgPool3d',
nn.AdaptiveMaxPool1d: 'AdaptiveMaxPool1d',
nn.AdaptiveMaxPool2d: 'AdaptiveMaxPool2d',
nn.AdaptiveMaxPool3d: 'AdaptiveMaxPool3d',
nn.Linear: 'Linear',
nn.Bilinear: 'Bilinear',
nn.Flatten: 'reshape',
nn.Unflatten: 'reshape',
nn.Embedding: 'Embedding',
nn.EmbeddingBag: 'EmbeddingBag',
nn.LSTM: 'LSTM',
nn.GRU: 'GRU',
nn.RNN: 'RNN',
nn.Upsample: 'Upsample',
nn.MultiheadAttention: 'MultiHeadAttention',
nn.LayerNorm: 'LayerNorm',
nn.GroupNorm: 'GroupNorm',
nn.InstanceNorm1d: 'InstanceNorm1d',
nn.InstanceNorm2d: 'InstanceNorm2d',
nn.InstanceNorm3d: 'InstanceNorm3d',

F.relu: 'ReLU',
F.relu6: 'ReLU6',
F.sigmoid: 'Sigmoid',
F.tanh: 'Tanh',
F.elu: 'ELU',
F.leaky_relu: 'LeakyReLU',
F.gelu: 'GELU',
F.selu: 'SELU',
F.prelu: 'PReLU',
F.softmax: 'Softmax',
F.log_softmax: 'LogSoftmax',
F.softplus: 'Softplus',
F.softsign: 'Softsign',
F.dropout: 'Dropout',
F.alpha_dropout: 'AlphaDropout',
F.max_pool1d: 'MaxPool1d',
F.max_pool2d: 'MaxPool2d',
F.max_pool3d: 'MaxPool3d',
F.avg_pool1d: 'AvgPool1d',
F.avg_pool2d: 'AvgPool2d',
F.avg_pool3d: 'AvgPool3d',
F.adaptive_avg_pool1d: 'AdaptiveAvgPool1d',
F.adaptive_avg_pool2d: 'AdaptiveAvgPool2d',
F.adaptive_avg_pool3d: 'AdaptiveAvgPool3d',
F.adaptive_max_pool1d: 'AdaptiveMaxPool1d',
F.adaptive_max_pool2d: 'AdaptiveMaxPool2d',
F.adaptive_max_pool3d: 'AdaptiveMaxPool3d',
F.batch_norm: 'BatchNorm',
F.instance_norm: 'InstanceNorm',
F.layer_norm: 'LayerNorm',
F.group_norm: 'GroupNorm',

```

```

F.linear: 'Linear',
F.bilinear: 'Bilinear',
F.conv1d: 'Conv1d',
F.conv2d: 'Conv2d',
F.conv3d: 'Conv3d',
F.conv_transpose1d: 'ConvTranspose1d',
F.conv_transpose2d: 'ConvTranspose2d',
F.conv_transpose3d: 'ConvTranspose3d',
F.embedding: 'Embedding',
F.embedding_bag: 'EmbeddingBag',
F.dropout2d: 'Dropout2d',
F.dropout3d: 'Dropout3d',
torch.flatten: 'reshape',
torch.cat: 'cat',
torch.stack: 'stack',
torch.add: 'add',
torch.sub: 'sub',
torch.mul: 'mul',
torch.div: 'div',
torch.mean: 'mean',
torch.sum: 'sum',
torch.reshape: 'reshape',
torch.transpose: 'reshape',
torch.permute: 'reshape',
torch.nn.functional.normalize: 'normalize',
torch.sin: 'sin',
torch.clamp: 'clamp',
torch.matmul: 'matmul',

'view': 'reshape',
'reshape': 'reshape',
'flatten': 'reshape',
'permute': 'reshape',
'transpose': 'reshape'
}

def count_fx_ops(model):
    f = io.StringIO()
    with redirect_stdout(f):
        traced = symbolic_trace(model)
        op_counter = Counter()

    for node in traced.graph.nodes:
        if node.op == 'call_module':
            mod = dict(traced.named_modules())[node.target]
            name = name_map.get(type(mod))
            if name:
                op_counter[name] += 1

        elif node.op in {'call_function', 'call_method'}:
            name = name_map.get(node.target)
            if name:
                op_counter[name] += 1

    return op_counter

def check_op_counts(actual_counts, required_counts):
    issues = []
    for key, required in required_counts.items():
        actual = actual_counts.get(key, 0)

```

```

if actual != required:
    issues.append(f"{key}: expected {required}, found {actual}")
for key in actual_counts:
    if key not in required_counts:
        issues.append(f"{key}: unexpected layer found (count = {actual_counts[key]})")
return issues

```

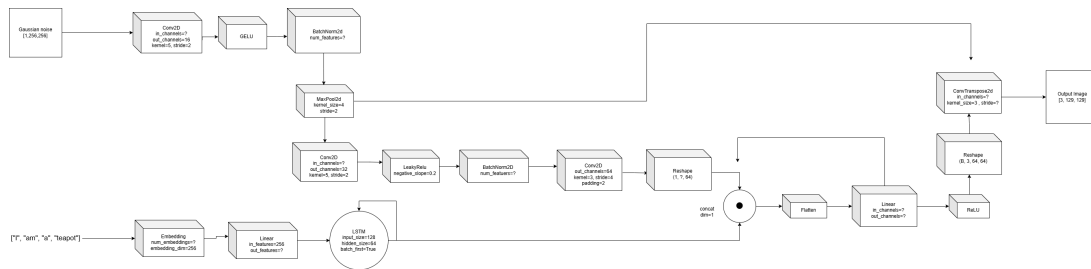
10.9. 1. Architecture

To better view the architecture diagram:

- **Right-click the image** and select “**Open image in a new tab**” to enable zoom for a clearer view.
- Alternatively, you can **download the image** using the link below:

[Download Architecture Diagram](#)

10.9.1. Diagram Preview:



Cell 0.4 (EDIT THIS CELL!)

```

class CustomModel(nn.Module):
    def __init__(self):
        super().__init__()

        self.top = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=5, stride=2),
            nn.GELU(),
            nn.BatchNorm2d(16),
            nn.MaxPool2d(kernel_size=4, stride=2),
            nn.Conv2d(16, 32, kernel_size=5, stride=2),
            nn.LeakyReLU(negative_slope=0.2),
            nn.BatchNorm2d(32),
            nn.Conv2d(32, 64, kernel_size=3, stride=4, padding=2)
        )

        self.bottom = nn.Sequential(
            nn.Embedding(100, 256),
            nn.Linear(256, 128),
            nn.LSTM(128, 64, batch_first=True),
        )

        self.tail = nn.Sequential(
            nn.Flatten(),
            nn.Linear(6144, 12288),
            nn.ReLU(),
        )

```

```

        self.tail2 = nn.Sequential(
            nn.ConvTranspose2d(3, 3, kernel_size=3, stride=2),
        )

    def forward(self, img, text):
        out = self.top(img)

        print(f"TOP BRANCH: {out.shape}")

        h, c = self.bottom(text)

        print(f"BOTTOM BRANCH: {h.shape}")

        out = out.reshape((1, -1, 64))
        print(f"RESHAPE BRANCH: {out.shape}")
        print(f"RESHAPE BRANCH: {h.shape}")
        combined = torch.cat((out, h), dim=1)

        print(f"FINAL BRANCH: {combined.shape}")

        combined = self.tail(combined)
        combined = combined.reshape((1, 3, 64, 64))
        combined = self.tail2(combined)

        print(f"FINAL BRANCH: {combined.shape}")

        return combined

```

Cell 1.2 (DO NOT EDIT THIS CELL!)

```

input_image = torch.randn(1, 1, 256, 256).to(device)
input_text = torch.randint(0, 100, (1, 32)).to(device)

custom_model = CustomModel().to(device)

output = custom_model(input_image, input_text)

try:
    actual_counts = count_fx_ops(custom_model)
    issues = check_op_counts(actual_counts, required_counts)

    if output.shape != (1, 3, 129, 129):
        raise AssertionError("Output shape is incorrect.")
    if issues:
        raise AssertionError("Operation count mismatch.")

    print("\n Congratulations! Your implementation is correct. You passed the first part of the exam! ")

except AssertionError as e:
    print(f"\n Error: {e}")
    if output.shape != (1, 3, 129, 129):
        print(f" - Expected output shape (1, 3, 129, 129), but got {output.shape}")
    if issues:
        print(" Mismatch in operation counts:")
        for issue in issues:
            print(" -", issue)

```

```
TOP BRANCH: torch.Size([1, 64, 8, 8])
BOTTOM BRANCH: torch.Size([1, 32, 64])
RESHAPE BRANCH: torch.Size([1, 64, 64])
RESHAPE BRANCH: torch.Size([1, 32, 64])
FINAL BRANCH: torch.Size([1, 96, 64])
FINAL BRANCH: torch.Size([1, 3, 129, 129])
```

Congratulations! Your implementation is correct. You passed the first part of the exam!

References

- [1] A. Krizhevsky, *Learning Multiple Layers of Features from Tiny Images*. Toronto, ON, Canada: University of Toronto, Technical Report, 2009. Available: <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>
- [2] M. Hodosh, P. Young, and J. Hockenmaier, “Framing Image Description as a Ranking Task: Data, Models and Evaluation Metrics,” *Journal of Artificial Intelligence Research*, vol. 47, pp. 853–899, 2013, doi: 10.1613/jair.3994. Available: <https://doi.org/10.1613/jair.3994>
- [3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An Image is Worth 16×16 Words: Transformers for Image Recognition at Scale,” in *Proceedings of the 9th International Conference on Learning Representations (ICLR)*, 2021. Available: <https://openreview.net/forum?id=YicbFdNTTy>
- [4] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, *Language Models are Unsupervised Multitask Learners*. San Francisco, CA, USA: OpenAI, Technical Report, 2019. Available: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
- [5] L. Fei-Fei, R. Fergus, and P. Perona, “One-Shot Learning of Object Categories,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 594–611, Apr. 2006, doi: 10.1109/TPAMI.2006.79. Available: <https://doi.org/10.1109/TPAMI.2006.79>
- [6] L. Bossard, M. Guillaumin, and L. Van Gool, “Food-101 – Mining Discriminative Components with Random Forests,” in *Computer Vision – ECCV 2014*, Lecture Notes in Computer Science, vol. 8694. Cham, Switzerland: Springer, 2014, pp. 446–461, doi: 10.1007/978-3-319-10599-4_29. Available: https://doi.org/10.1007/978-3-319-10599-4_29
- [7] C. Rashtchian, P. Young, M. Hodosh, and J. Hockenmaier, “Collecting Image Annotations Using Amazon’s Mechanical Turk,” in *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon’s Mechanical Turk*. Los Angeles, CA, USA: Association for Computational Linguistics, 2010, pp. 139–147. Available: <https://aclanthology.org/W10-0721/>
- [8] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft COCO: Common Objects in Context,” in *Computer Vision – ECCV 2014*, Lecture Notes in Computer Science, vol. 8693. Cham, Switzerland: Springer, 2014, pp. 740–755, doi: 10.1007/978-3-319-10602-1_48. Available: https://doi.org/10.1007/978-3-319-10602-1_48

Dataset and Model Attribution

CIFAR-10 — Krizhevsky [1]. Mentioned as an optional dataset in the image-colorization assignment. The dataset is not distributed with this volume, and no CIFAR-10 images are reproduced in it.

Source: <https://www.cs.toronto.edu/~kriz/cifar.html>

Caltech101 — Fei-Fei, Fergus, and Perona [5]. Mentioned as an optional dataset in the image-colorization assignment. The dataset is not distributed with this volume, and no Caltech101

images are reproduced in it.

Dataset source: <https://data.caltech.edu/records/mzrjq-6wc02>

Dataset DOI: <https://doi.org/10.22002/D1.20086>

Food-101 — Bossard, Guillaumin, and Van Gool [6]. Mentioned as an optional dataset in the image-colorization assignment. The dataset is not distributed with this volume, and no Food-101 images are reproduced in it.

Dataset source: https://data.vision.ee.ethz.ch/cvl/datasets_extra/food-101/

Flickr8k — Rashtchian et al. [7] and Hodosh, Young, and Hockenmaier [2]. Used in the image-captioning assignment. The dataset itself is not distributed with this volume. The example photographs reproduced in Chapter 3 are independently licensed images and are not Flickr8k photographs. The Flickr8k dataset page states that its image captions are released under a Creative Commons Attribution-ShareAlike license.

Dataset source: <https://hockenmaier.cs.illinois.edu/8k-pictures.html>

MS COCO — Lin et al. [8]. MS COCO is not directly used or distributed as part of this volume. It is listed here as training-data provenance associated with the pretrained image-captioning model used for the example prediction in Chapter 3.

Dataset source: <https://cocodataset.org/>

nlpconnect/vit-gpt2-image-captioning — pretrained image-captioning model used to generate the example predicted caption shown in Chapter 3. The model is distributed on the Hugging Face Model Hub under the Apache License 2.0. Its model card identifies it as a PyTorch version of the ydshieh/vit-gpt2-coco-en-ckpt's image-captioning checkpoint. The model files are not redistributed with this volume.

Model source: <https://huggingface.co/nlpconnect/vit-gpt2-image-captioning>

Model DOI: <https://doi.org/10.57967/hf/0222>

License: <https://www.apache.org/licenses/LICENSE-2.0>

Original checkpoint: <https://huggingface.co/ydshieh/vit-gpt2-coco-en-ckpt>

Image Credits

The example photographs used in the image-captioning examples are independently licensed images obtained via Wikimedia Commons. Images were resized and, where applicable, cropped for presentation.

- *Tabby cat with blue eyes-3336579* — https://commons.wikimedia.org/wiki/File:Tabby_cat_with_blue_eyes-3336579.jpg — AdinaVoicu, via Wikimedia Commons (originally Pixabay), CC0 1.0.
License: <https://creativecommons.org/publicdomain/zero/1.0/>
- *Portrait of a labrador retriever* — https://commons.wikimedia.org/wiki/File:Portrait_of_a_labrador_retriever.jpg — Dktue, via Wikimedia Commons, CC0 1.0.
License: <https://creativecommons.org/publicdomain/zero/1.0/>
- *Komondor am Sylter Strand – unser Suchbild....)* — https://commons.wikimedia.org/wiki/File:Komondor_am_Sylter_Strand_-_unser_Suchbild....%29.jpg — Wydra22, via Wikimedia Commons, CC BY-SA 4.0.
License: <https://creativecommons.org/licenses/by-sa/4.0/>

- *Zebras grazing among acacia trees, Serengeti National Park* — [https://commons.wikimedia.org/wiki/File:Zebras grazing among acacia trees, Serengeti National Park.jpg](https://commons.wikimedia.org/wiki/File:Zebras_grazing_among_acacia_trees,_Serengeti_National_Park.jpg) — Agness Abubakar Saidi, via Wikimedia Commons, CC0 1.0.
License: <https://creativecommons.org/publicdomain/zero/1.0/>
- *African elephant walking* — [https://commons.wikimedia.org/wiki/File:African elephant walking.jpg](https://commons.wikimedia.org/wiki/File:African_elephant_walking.jpg) — Stephen Jacobs (Why Steve), via Wikimedia Commons, CC BY 4.0.
License: <https://creativecommons.org/licenses/by/4.0/>
- *2023 photo – People are walking through the station hall or shopping mall behind Central station, Amsterdam city. An urban image of modern interior design of a Mirror Palace. Street photography by Fons Heijnsbroek* — [https://commons.wikimedia.org/wiki/File:2023 photo - People are walking through the station hall or shopping mall behind Central station, Amsterdam city. An urban image of modern interior design of a Mirror Palace. Street photography by Fons Heijnsbroek](https://commons.wikimedia.org/wiki/File:2023_photo_-_People_are_walking_through_the_station_hall_or_shopping_mall_behind_Central_station,_Amsterdam.jpg) — Fons Heijnsbroek, via Wikimedia Commons, CC0 1.0.
License: <https://creativecommons.org/publicdomain/zero/1.0/>
- *Traffic Jam – Eastern Metropolitan Bypass – Science City Area – Kolkata 2015-12-31 8335* — [https://commons.wikimedia.org/wiki/File:Traffic Jam - Eastern Metropolitan Bypass - Science City Area - Kolkata 2015-12-31 8335.JPG](https://commons.wikimedia.org/wiki/File:Traffic_Jam_-_Eastern_Metropolitan_Bypass_-_Science_City_Area_-_Kolkata_2015-12-31_8335.JPG) — Biswarup Ganguly, via Wikimedia Commons, CC BY 3.0.
License: <https://creativecommons.org/licenses/by/3.0/>
- *Club Big Breakfast, The Coffee Club Riverton, 2026 (02)* — [https://commons.wikimedia.org/wiki/File:Club Big Breakfast, The Coffee Club Riverton, 2026 %2802%29.jpg](https://commons.wikimedia.org/wiki/File:Club_Big_Breakfast,_The_Coffee_Club_Riverton,_2026_%2802%29.jpg) — Bahnfreund, via Wikimedia Commons, CC BY-SA 4.0.
License: <https://creativecommons.org/licenses/by-sa/4.0/>

The image-colorization illustration in Chapter 1 and the encoder–attention–decoder schematic in Chapter 3 are original figures created by the authors of this volume.

Where Creative Commons material has been cropped or resized, the modification is limited to presentation. The original creators, source pages, and applicable licenses are identified above.