Master in Computer Vision *Barcelona*

Module:	C5 – Task 3
Project:	**Object Detection**
Coordinator:	E. Valveny
Team 8:	G. Grigoryan,V. Heuer, P. Zetterberg

Task 1: Download and Split the Data

Entry 1(idx = 0): „Miso-Butter Roast Chicken With Acorn Squash Panzanella"

Dataset Metrics		
parameter	number images	relative
total images	13,501	1.00
train split	10,800	0.80
valid split	1,350	0.10
test split	1,351	0.10

Method

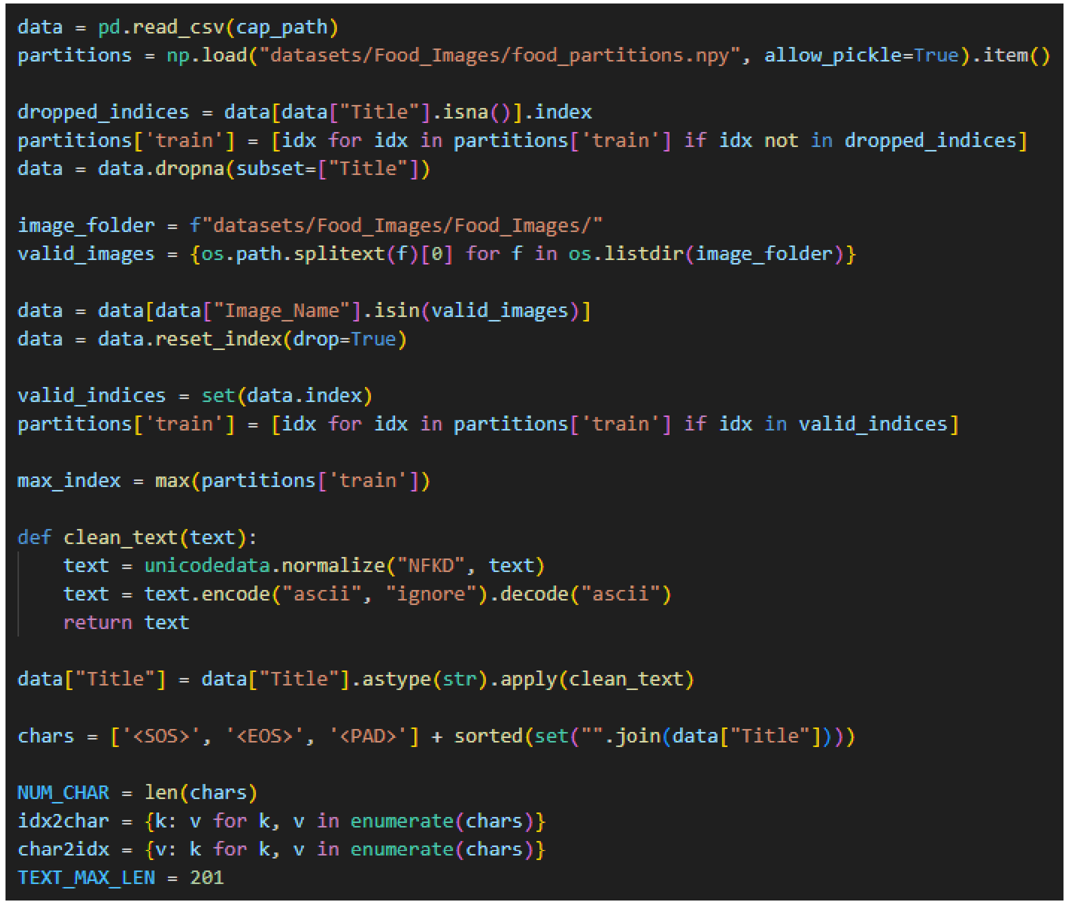
- We downloaded the dataset from kaggle (https://www.kaggle.com/datasets/pes12017000148/food-ingredients-and-recipe-dataset-with-images/data)
- We split the data to 80 % training, 10 % validation and 10 % test by loading the annotation file with pandas and using „sk.learn.model_selection.train_test_split()" function, this splits the idexes in random order into the desired precentages which can be stored in a dictionary.
- We saved the dictionary as .npy file to later use in our main script.
- The website states that the dataset contains 13,582 images in total. The csv file only contains 13,501 unique entries.

Task 2: Dataloader for Dataset with Recipe Titles as GT

We deciced to use the data class from the baseline, therefore, we had to preprocess the data.

Code Explaination

- After loading the data the same way as in the baseline code a few adjustments have to be made.
- First we remove entries without title while updating the „train“ partition.
- Then we compare the data with the image folder to remove missing images from data.
- We update the partitions after filtering.
- We reove non-ASCII characters from „Title“ to prevent issues with special characters.
- We extract the unique characters in „Title“ while adding the SOS, EOS and PAD characters.



```python
data = pd.read_csv(cap_path)
partitions = np.load("datasets/Food_Images/food_partitions.npy", allow_pickle=True).item()

dropped_indices = data[data["Title"].isna()].index
partitions['train'] = [idx for idx in partitions['train'] if idx not in dropped_indices]
data = data.dropna(subset=["Title"])

image_folder = f"datasets/Food_Images/Food_Images/"
valid_images = {os.path.splitext(f)[0] for f in os.listdir(image_folder)}

data = data[data["Image_Name"].isin(valid_images)]
data = data.reset_index(drop=True)

valid_indices = set(data.index)
partitions['train'] = [idx for idx in partitions['train'] if idx in valid_indices]

max_index = max(partitions['train'])

def clean_text(text):
    text = unicodedata.normalize("NFKD", text)
    text = text.encode("ascii", "ignore").decode("ascii")
    return text

data["Title"] = data["Title"].astype(str).apply(clean_text)

chars = ['<SOS>', '<EOS>', '<PAD>'] + sorted(set("".join(data["Title"])))

NUM_CHAR = len(chars)
idx2char = {k: v for k, v in enumerate(chars)}
char2idx = {v: k for k, v in enumerate(chars)}
TEXT_MAX_LEN = 201
```

1: https://datasetninja.com/kitti-mots

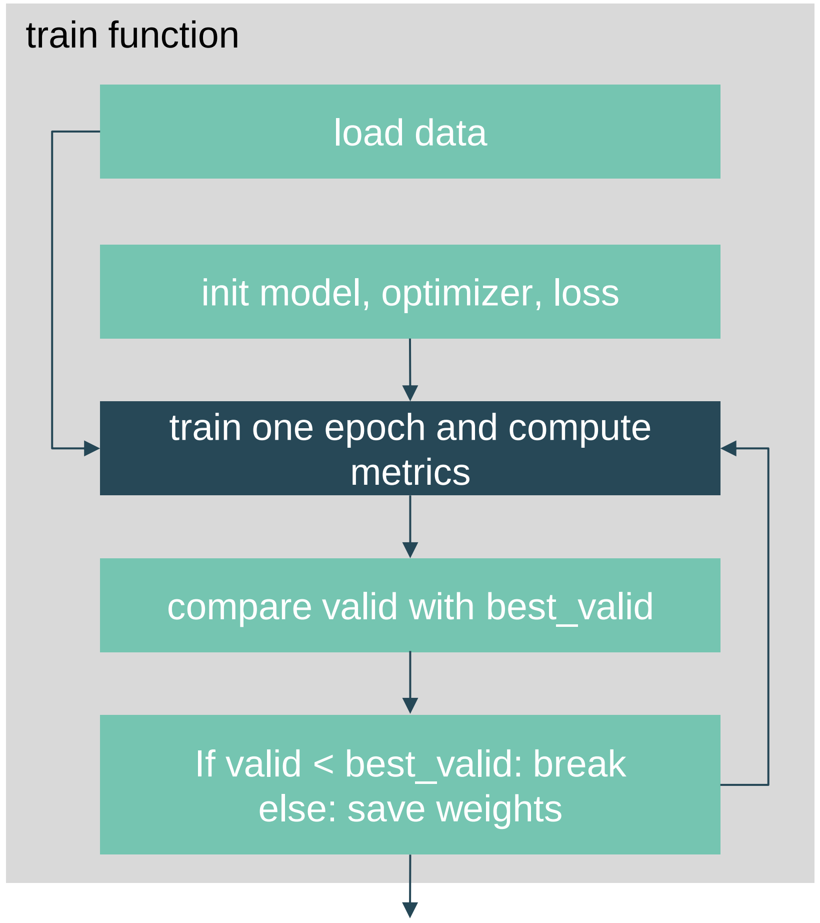
Task 3/4.1: Train and Evaluate the Baseline Model – Experimental Setup.

Experimental Setup

- For training the model we decided to split the functionality into train and into train_one_epoch.
- This allows us to controll the data loading, logging, validation and early stopping seperate from the actual training.
- We used wandb for logging the metrics.
- We decided to use following hyperparameters:

Optimizer:	Adam
Lr:	1e-3
Batch Size:	8
Loss:	Cross Entropy Loss
Epochs:	10

- Also we implemented a teacher forcing method that with a value of 0.5.



train function
load data
init model, optimizer, loss
train one epoch and compute metrics
compare valid with best_valid
If valid < best_valid: break
else: save weights

Task 3/4: Train and Evaluate the Baseline Model – Results.

Two very different dishes with the same prediction: „<SOS>Coase "

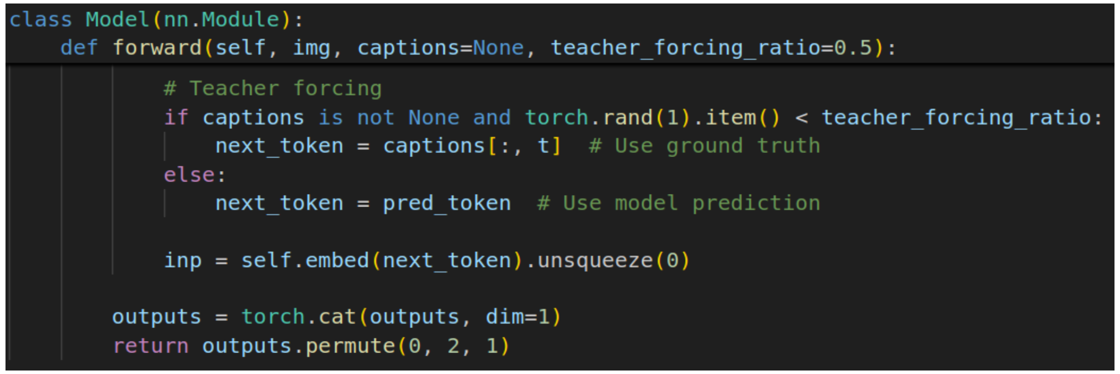
Results of Baseline Model

metric	before training	10 epochs
BLEU-1	0.0001	0.0012
BLEU-2	0.0000	0.0000
ROUGE-L	0.0000	0.0024
METEOR	0.0012	0.0012
training time		~ 8 hrs.

Analysis

Even though we used a RTX3090 for training, the traing times remained significant. This made it difficult to go for an exstensive parameter search. The model has shown some improvement in generating captions, as evidenced by the slight increase in BLEU-1 and ROUGE-L scores after 10 epochs. However, the BLEU-2 and METEOR scores remained unchanged, indicating that the model is still struggling to generate coherent bigrams and semantically accurate captions. While BLEU-1 improvement suggests the model is capturing basic word patterns, it still lacks the ability to generate meaningful word combinations. The ROUGE-L score improvement indicates the model is beginning to capture some relevant subsequences, but overall, the performance is still quite low. To improve, further hyperparameter tuning, a longer training period, and more advanced architectures, such as attention mechanisms, could help boost the model's performance. The qualitative analysis shows that the model is probably underfitting. It does not seem to generalize well. Most of the predictions are either full of tokens or start with "Coa" without any real words.

Task 6.1: Changing Encoder to VGG19



```
class Model(nn.Module):
    def forward(self, img, captions=None, teacher_forcing_ratio=0.5):
            # Teacher forcing
            if captions is not None and torch.rand(1).item() < teacher_forcing_ratio:
                next_token = captions[:, t]  # Use ground truth
            else:
                next_token = pred_token  # Use model prediction

            inp = self.embed(next_token).unsqueeze(0)

        outputs = torch.cat(outputs, dim=1)
        return outputs.permute(0, 2, 1)
```

Teacher Forcing

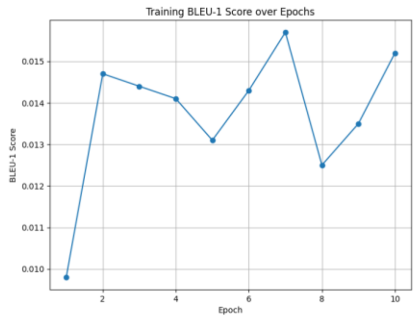
Results of VGG19 Model on the test set		
metric	before training	after 10 epochs
BLEU-1	0.0017	0.0007
BLEU-2	0.0007	0.0000
ROUGE-L	0.0011	0.0007
METEOR	0.0026	0.0004
training time		~ 8 hrs.

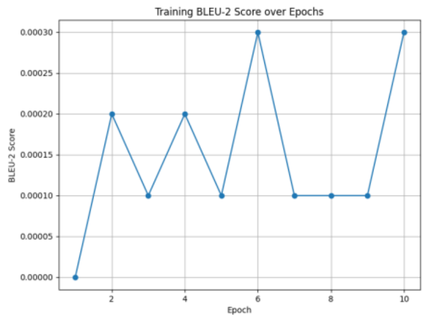
Method

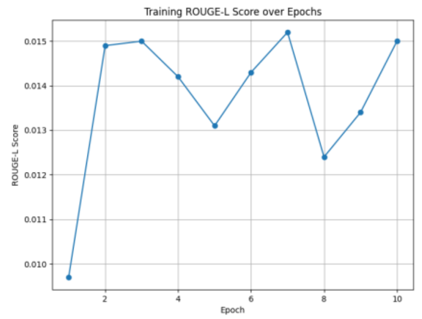
- To investigate the influence of the encoder, we trained the model with the same parameters as before, but we changed the encoder from ResNet to VGG19.
- Altering the code to achieve this was releatively straight forward with only a few changes in the model class.
- VGG19 was used from the transformers library for this training.
- The model performed out of the box better than the ResNet model.
- Nevertheless the training time was very long so we implemented a trainer which sped up the model by a factor of 4 (from 8 to 2 hours per training – 10 epochs)
- Teacher Forcing: during training, the model learns to predict the next token based on the previously predicted tokens, but instead of always using the model's own predictions, we sometimes feed it the true previous word (ground truth).
- The training led to very low values for every metric, indicating a problem with the training.
- Our conclusion was that the model we used from vgg19 was basicly learning from scratch again (with only few epochs) in the way we implemented it, therefore it couldn't get any higher values in the metrics.

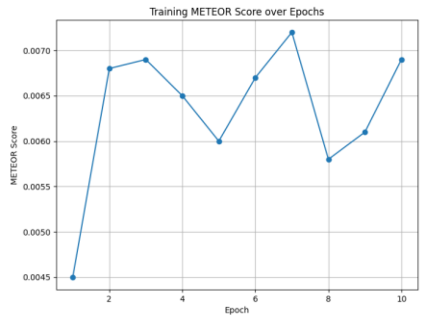
➢ Results?

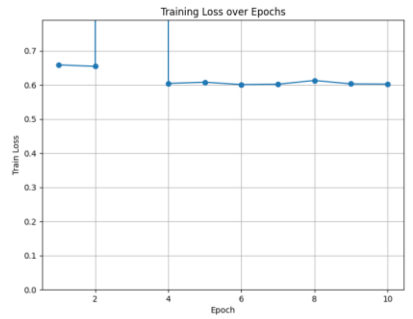
Task 6.1: Changing Encoder to VGG19





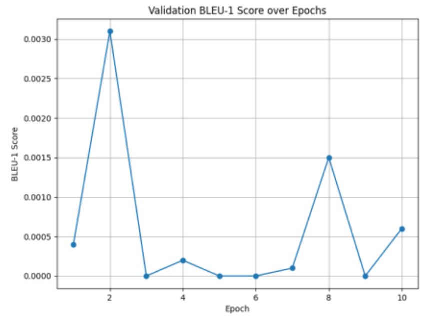


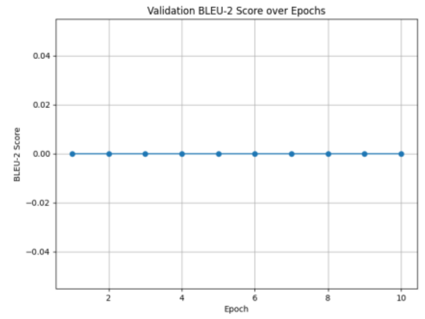


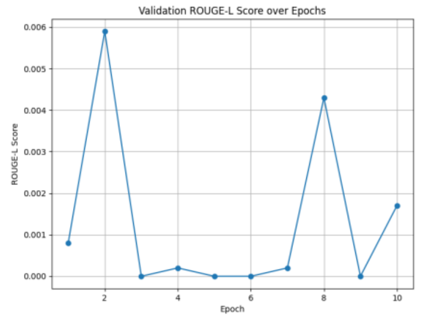


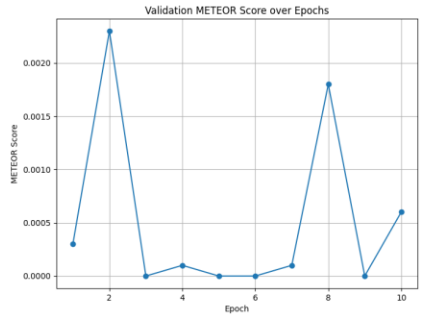
Metrics

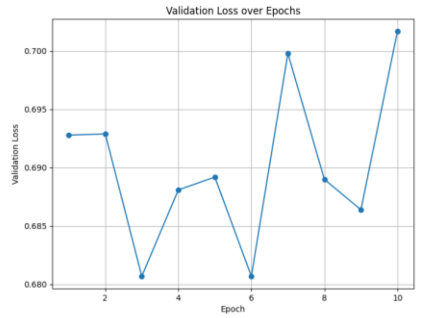
Loss







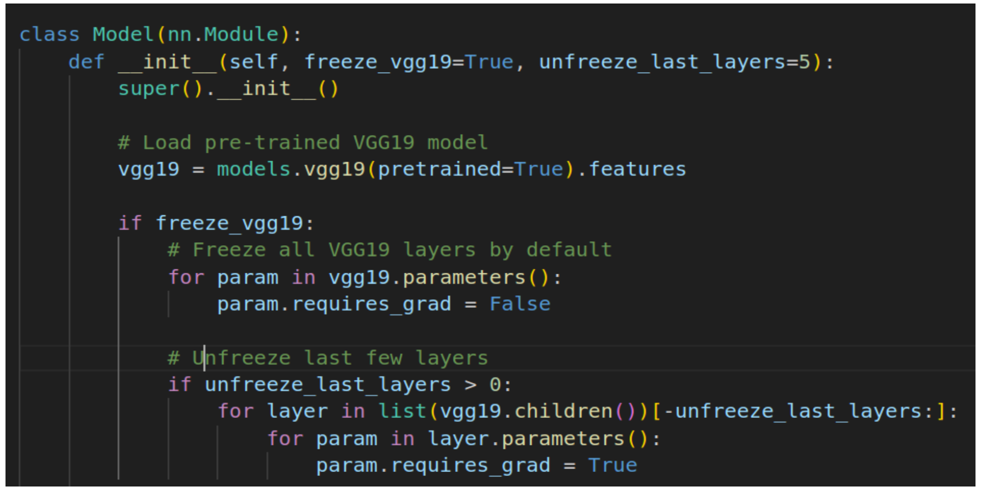




Observations & Solution

- ➢ The 5 graphs in the top are the metrcis and the loss function evaluated from every epoch in training, while the 5 graphs below are metrics and loss for the validation in each epoch. Clearly something went wrong here.
- ➢ The metrics remained extremely low and nearly flat, which suggested the model wasn't learning properly.
- ➢ The training loss was extremly high at one point (324.0441), while the validation loss stayed relatively low, indicating some instability in training.
- ➢ The model was likely trying to learn from scratch, instead of using the pretrained knowledge from VGG19.
- ➢ This could have happened because the feature extraction layers of VGG19 were not frozen and the model wasn't using the pretrained features effectively, especially the low-level features that are crucial for image recognition.

Task 6.1: Changing Encoder to VGG19 with frozen layers



```
class Model(nn.Module):
    def __init__(self, freeze_vgg19=True, unfreeze_last_layers=5):
        super().__init__()

        # Load pre-trained VGG19 model
        vgg19 = models.vgg19(pretrained=True).features

        if freeze_vgg19:
            # Freeze all VGG19 layers by default
            for param in vgg19.parameters():
                param.requires_grad = False

            # Unfreeze last few layers
            if unfreeze_last_layers > 0:
                for layer in list(vgg19.children())[-unfreeze_last_layers:]:
                    for param in layer.parameters():
                        param.requires_grad = True
```

Freezing Layers

Results of VGG19 Model (frozen layers) on the test set

metric	before training	Best model after 10 epochs
BLEU-1	0.0017	0.0376
BLEU-2	0.0007	0.0000
ROUGE-L	0.0011	0.0414
METEOR	0.0026	0.0182
training time		~ 1,5 hrs.

Method

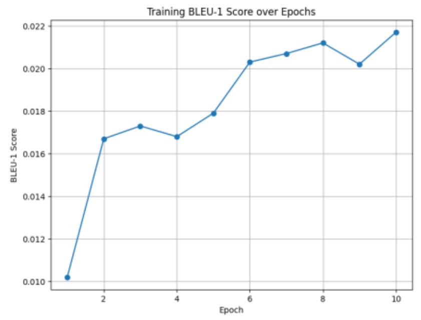
- ➢ The idea here is to use the knowledge the VGG19 model already has from being pretrained on a massive dataset (ImageNet).
- ➢ Instead of training the entire model from scratch, we freeze the early layers, which are responsible for detecting basic image features like edges, shapes, and textures.
- ➢ Only the last few layers are left trainable.
- ➢ These layers are more specialized and are responsible for understanding more complex patterns and making final predictions.

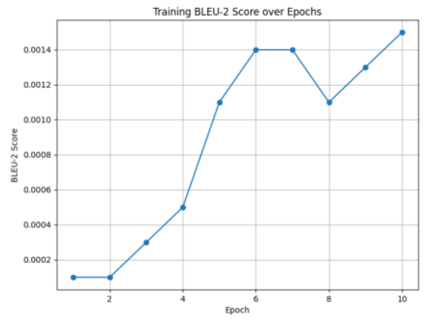
Why This is Beneficial:

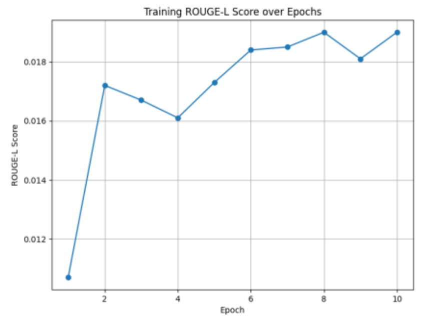
- ➢ Reduces Training Time: Since most of the model is kept frozen, only a small part of the network is updated, making training faster.
- ➢ Improves Stability: Keeping the reliable, pretrained features intact ensures the model should help prevent overfit or diverge during training.
- ➢ Makes Fine-Tuning Possible: The model only needs to learn the specific features relevant to our dataset instead of starting from scratch.

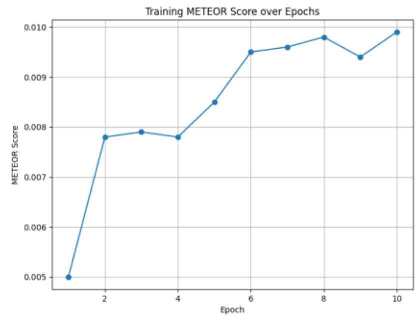
➢ Results?

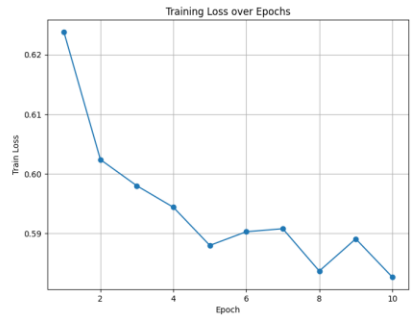
Task 6.1: Changing Encoder to VGG19 with frozen layers





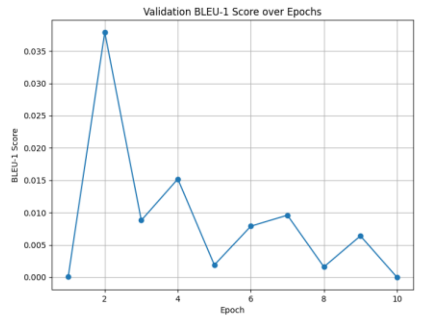


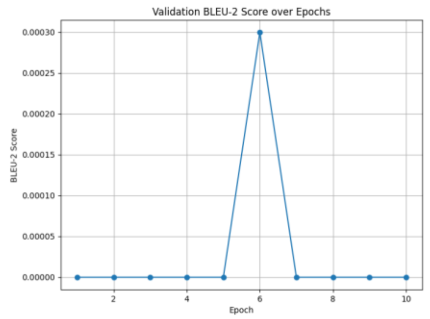


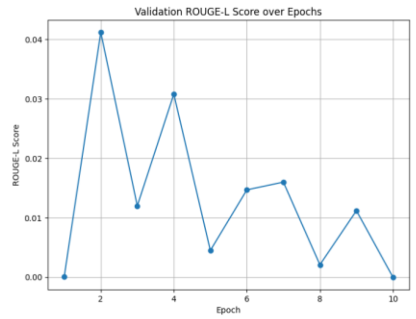


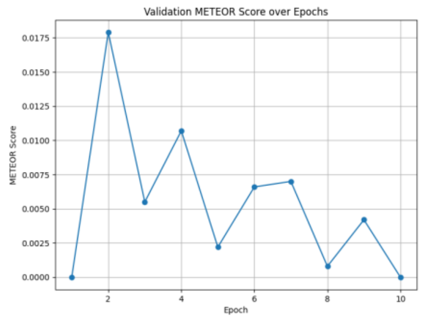
Metrics

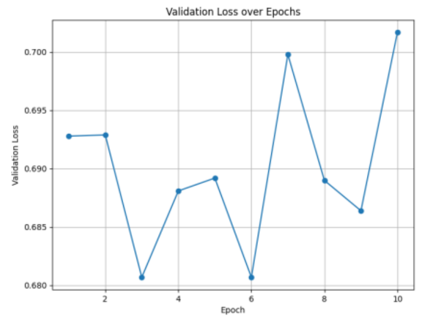
Loss







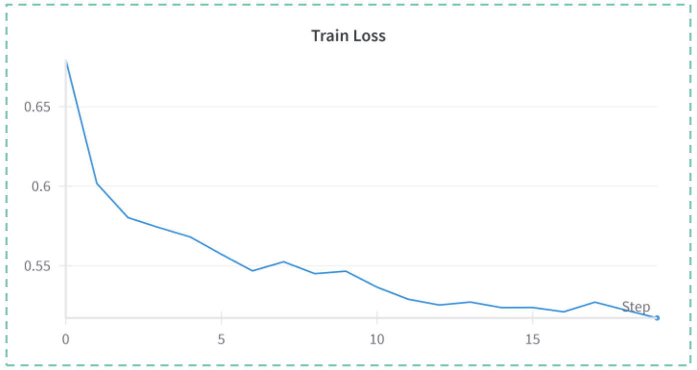




Observations

- ➢ The 5 graphs in the top are the metrcis and the loss function evaluated from every epoch in training, while the 5 graphs below are metrics and loss for the validation in each epoch.
- ➢ Compared to the previous attempt, the metrics during training are steadily increasing, and the loss values are decreasing, which clearly indicates successful learning.
- ➢ However, there's a noticeable pattern of increasing training metrics but decreasing validation metrics over the epochs, which combined with the steadily decreasing training loss and fluctuating validation loss, suggests overfitting.
- ➢ The model is learning to fit the training data well, but it's struggling to generalize to the validation data.
- ➢ Despite the overfitting signs and room for improvements, the overall performance is much better than before, and the model's ability to learn from the frozen VGG19 layers proves that leveraging pretrained weights was the right choice.

Task 6.2: Changing Decoder to LSTM



Results of LSTM Model on the test set		
metric	before training	10 epochs
BLEU-1	0.0004	0.0019
BLEU-2	0.0000	0.0003
ROUGE-L	0.0010	0.0042
METEOR	0.0011	0.0023
training time		~ 8 hrs.

Method

- To investigate the influence of the decoder, we trained the model with the same parameters as before, but we changed the decoder from GRU to LSTM. Altering the code to achieve this was releatively straigt forward with only a few changes in the model class. We used LSTM from the pytorch library.
- The model performed out of the box a little better than the baseline model but worse than the VGG19 model.
- The change from GRU to LSTM led to a noticeable improvement in BLEU-1 and ROUGE-L scores, suggesting better word-level and subsequence matching. While BLEU-2 and METEOR scores still show minimal improvement, the LSTM seems to help capture some longer-range dependencies and overall context. These results indicate that the LSTM architecture might be better at handling the sequence generation, but further tuning and more training could be needed to significantly improve performance.

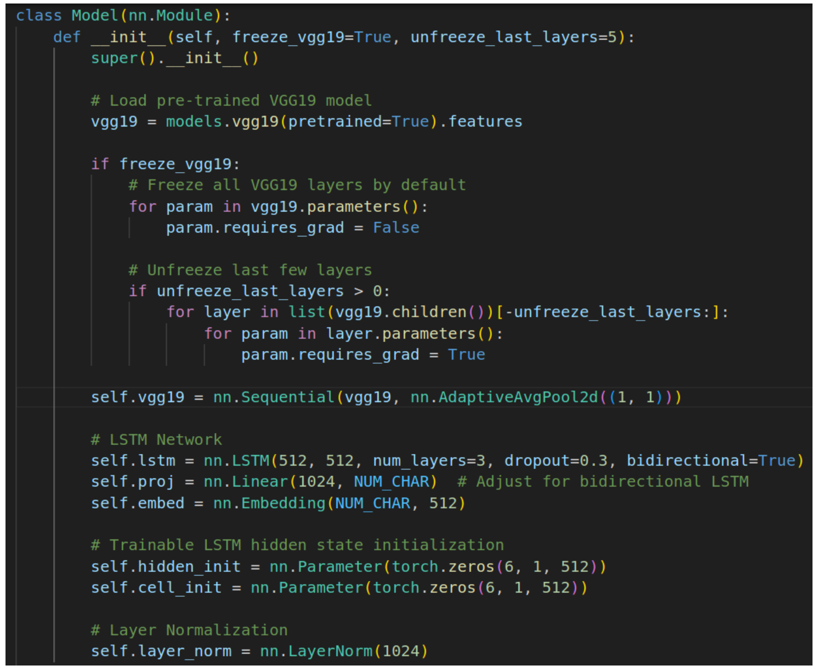
Task 6.3: Changing Text Representation Level

Method

- We were not able to successfully train a word-level model in time. We are in the process of training these but will prbably not be able to deliver results on time.

➢ Results?

Task 6.4: VGG19 and LSTM Combined



```
class Model(nn.Module):
    def __init__(self, freeze_vgg19=True, unfreeze_last_layers=5):
        super().__init__()

        # Load pre-trained VGG19 model
        vgg19 = models.vgg19(pretrained=True).features

        if freeze_vgg19:
            # Freeze all VGG19 layers by default
            for param in vgg19.parameters():
                param.requires_grad = False

            # Unfreeze last few layers
            if unfreeze_last_layers > 0:
                for layer in list(vgg19.children())[-unfreeze_last_layers:]:
                    for param in layer.parameters():
                        param.requires_grad = True

        self.vgg19 = nn.Sequential(vgg19, nn.AdaptiveAvgPool2d((1, 1)))

        # LSTM Network
        self.lstm = nn.LSTM(512, 512, num_layers=3, dropout=0.3, bidirectional=True)
        self.proj = nn.Linear(1024, NUM_CHAR)  # Adjust for bidirectional LSTM
        self.embed = nn.Embedding(NUM_CHAR, 512)

        # Trainable LSTM hidden state initialization
        self.hidden_init = nn.Parameter(torch.zeros(6, 1, 512))
        self.cell_init = nn.Parameter(torch.zeros(6, 1, 512))

        # Layer Normalization
        self.layer_norm = nn.LayerNorm(1024)
```

Results of VGG19 + LSTM Model on the test set

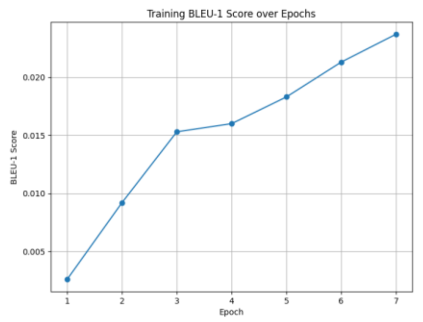
metric	before training	Best model after 7epochs
BLEU-1	0.0000	0.0000
BLEU-2	0.0000	0.0000
ROUGE-L	0.0000	0.0000
METEOR	0.0000	0.0000
training time		~ 2,5 hrs. (so ~ 3,5 hrs for 10 epochs)

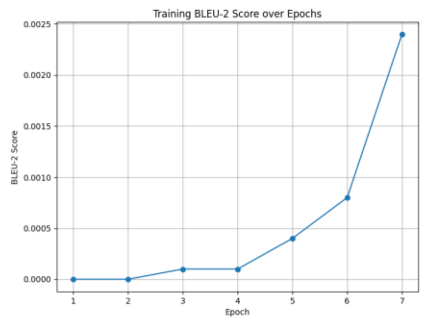
Method

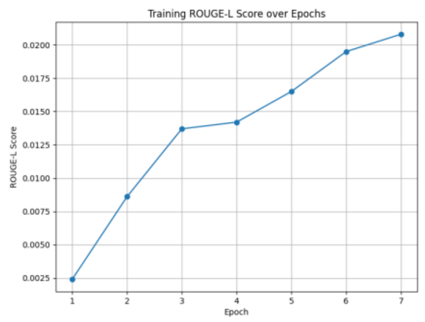
- Freezing the early layers of VGG19 allows it to act as a feature extractor, identifying basic features like edges and shapes.
- These features are then passed to a Bidirectional LSTM, which reads them both forward and backward, potentially capturing more context from the image.
- This method could have improved the model's understanding of the image and leads to better generalization.
- But the validation and test results show, that it's not the case or that something went wrong in the training
- Even though our training curves look fine,l the metrics all yield 0.
- When reviewing some predictions the results seem okay. They tend to start with the letter „S“. The apparent variance of the predictions is higher compared to the previous models. The captions are still not meaningful, but multiple words.

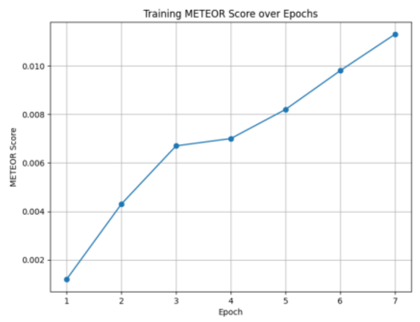
- Results?

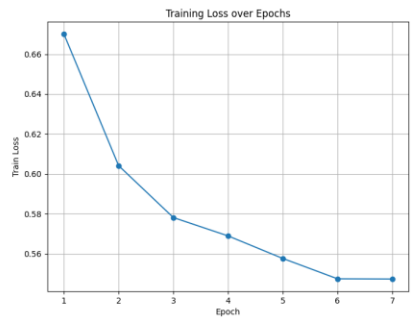
Task 6.4: VGG19 and LSTM Combined





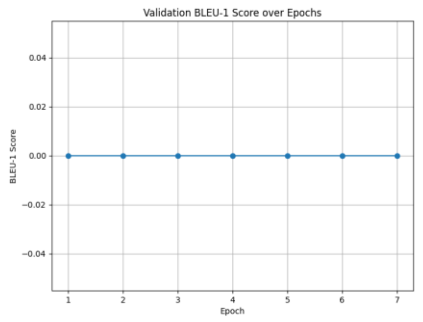


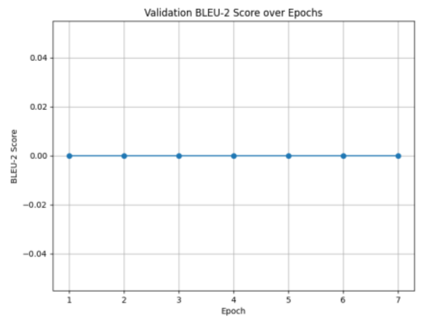


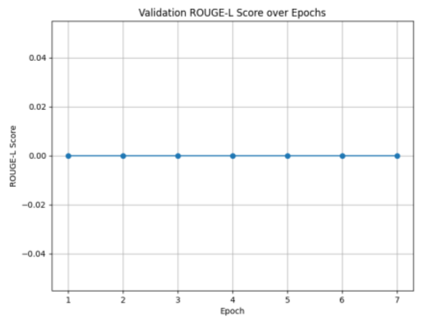


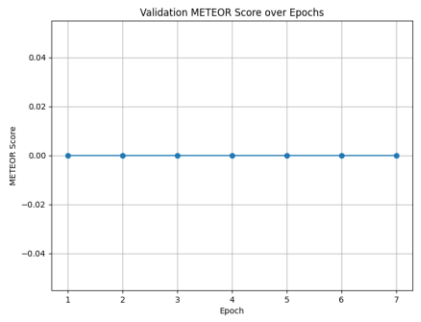
Metrics

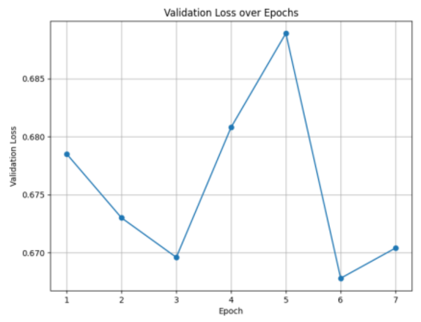
Loss







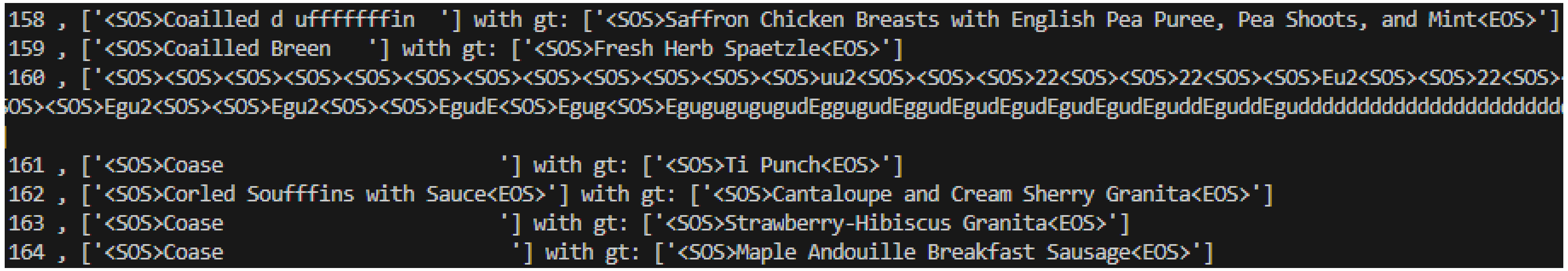




Conclusion

- The training results for the VGG19-BiLSTM model show a consistent improvement in BLEU-1, BLEU-2, ROUGE-L, and METEOR scores over the epochs, indicating that the model is learning during training.
- Additionally, the continuous decrease in training loss further supports successful training.
- However, all validation metrics (BLEU-1, BLEU-2, ROUGE-L, METEOR) remain at zero throughout the epochs, suggesting that something is fundamentally wrong with the validation process.
- This issue could be caused by a mistake in the validation loop, incorrect evaluation metric implementation, or a mismatch between training and validation preprocessing steps.

Qualitative Analysis



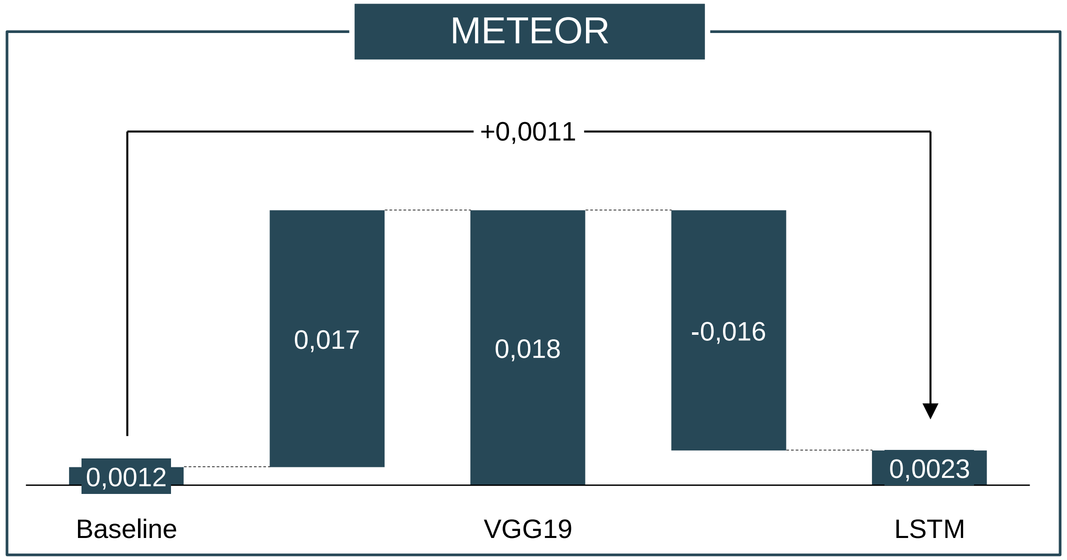
```
158 , ['<SOS>Coailled d ufffffffin  '] with gt: ['<SOS>Saffron Chicken Breasts with English Pea Puree, Pea Shoots, and Mint<EOS>']
159 , ['<SOS>Coailled Breen   '] with gt: ['<SOS>Fresh Herb Spaetzle<EOS>']
160 , ['<SOS><SOS><SOS><SOS><SOS><SOS><SOS><SOS><SOS><SOS><SOS><SOS>uu2<SOS><SOS><SOS>22<SOS><SOS>22<SOS><SOS>Eu2<SOS><SOS>22<SOS>
OS><SOS>Egu2<SOS><SOS>Egu2<SOS><SOS>EgudE<SOS>Egug<SOS>EgugugugugudEggugudEggudEgudEgudEgudEgudEguddEguddEguddddddddddddddddddddd

161 , ['<SOS>Coase                        '] with gt: ['<SOS>Ti Punch<EOS>']
162 , ['<SOS>Corled Soufffins with Sauce<EOS>'] with gt: ['<SOS>Cantaloupe and Cream Sherry Granita<EOS>']
163 , ['<SOS>Coase                        '] with gt: ['<SOS>Strawberry-Hibiscus Granita<EOS>']
164 , ['<SOS>Coase                         '] with gt: ['<SOS>Maple Andouille Breakfast Sausage<EOS>']
```

Analysis

- When reviewing the outputs of the trained baseline we can see that the models is often beginning the caption with „Coa“. Fruthermore, the model is not capable of producing meaningful captions. When reviewing the outputs, most of them look like output with index 160. This might be because of a problem in handling the tokenization. We think that these problems point to undefitting. Becuase of the intense computation, we were not able to train the models for a lot longer than 10 epochs.
- When reviewing the VGG19 epochs with frozen layers, the first one also has the tendency to predict „Coa“ and „Cos“. Here there are almost no meaninigful and mostly one worded predictions. As we progress through the epochs the captions suddenly all start with „S“ and are mostly the same or similiar 6 letter words. But there appear longer captions with multiple words. A few real words are predicted, they do not match the gt.

➢ Results?

Team 8: Despite extensively training our models and experimenting with various techniques, we were unable to generate meaningful captions, highlighting the challenges in achieving coherent and accurate text generation.



Performance of Different Models

metric	Baseline	VGG19	LSTM
BLEU-1	0.0012	0.0376	0.0019
BLEU-2	0.0000	0.0000	0.0003
ROUGE-L	0.0024	0.0414	0.0042
METEOR	0.0012	0.0182	0.0023

Two very different dishes with the same prediction: „<SOS>Coase “

Baseline	‘<SOS>Coase ‘ ‘<SOS>Corled Soufffins with Sauce<EOS>‘ ‘<SOS>Coailled Breen ‘ ‘<SOS><SOS>…Egu2<SOS><SOS><SOS><SOS> ‘
VGG19	‘<SOS>Saasted a‘ ‘<SOS>Shice‘ ‘<SOS>Saae‘ ‘<SOS >Shesteeand Shorted Potato and ‘