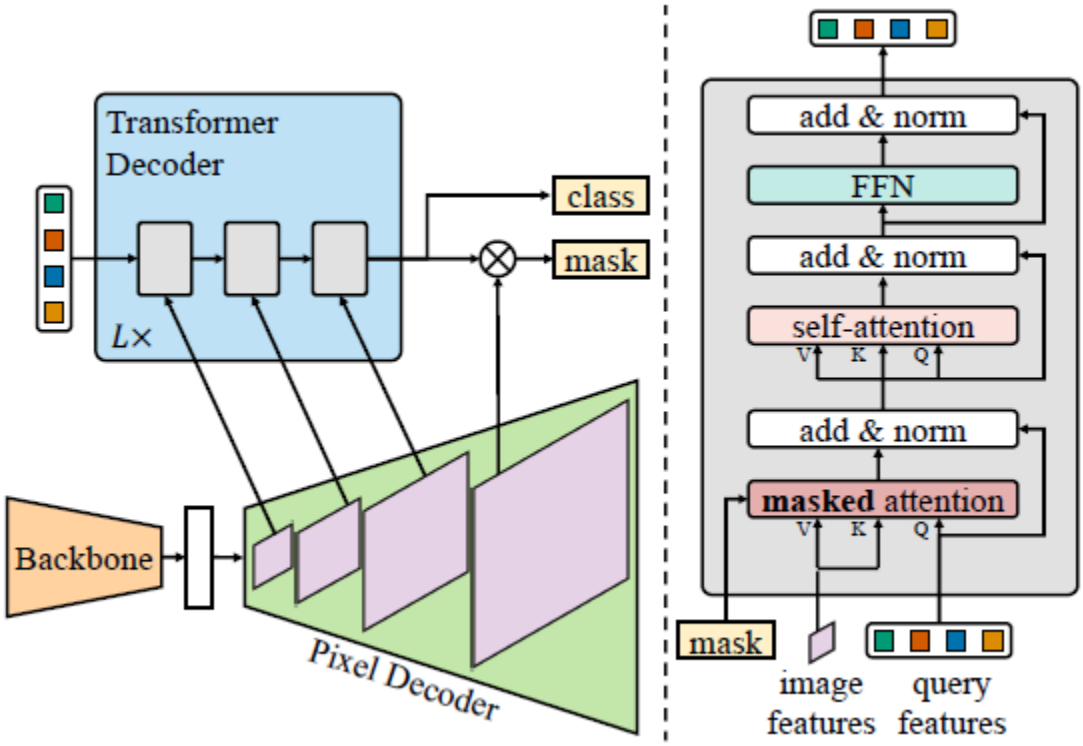
Arhitektura Mask2Formera



Slika 1. Izgled arhitekture

- Izgled citave arhitekture, u kojoj su 3 kljucna dijela: *Backbone, Pixel Decoder, Transformer Decoder.*

- Za opis arhitekture koristen je *config* fajl koji definise dijelove koji su iskoristeni za pravljenje modela, a koji se nalazi na putanji: https://github.com/open-mmlab/mmsegmentation/blob/main/configs/mask2former/mask2former_r50_8xb2-160k_ade20k-512x512.py. [1]
- Konkretno ovaj *config* fajl je definisao model koji koristi *ResNet50* kao *backbone* treniran na slikama iz baze *ADE20K*, i slike su pretvorene u velicinu 512x512.

Backbone

- U navedenom *config* fajlu na linijama 19-28 (backbone=dict(type='ResNet') definisano je da se kao *backbone* koristi *ResNet50.*
- Arhitektura *ResNet50* se moze naci na putanji: https://github.com/open-mmlab/mmsegmentation/blob/main/mmseg/models/backbones/resnet.py. [2]
- Uzimajuci u obzir da ovo nije sustina rada koji je napisan za ovaj model i to sto mogu razlicite arhitekture da se koriste kao *backbone* nisam previse ulazio u detalje implementacije *ResNet50*.

Decode Head

- U linijama 29-136 (decode_head=dict(type='Mask2FormerHead')) u *config* fajlu je definisan ostatak modela.
- Prva stvar koja se navodi kao *decode head* jeste klasa *Mask2FormerHead* koja se nalazi na putanji: https://github.com/open-mmlab/mmsegmentation/blob/main/mmseg/models/decode_heads/mask2former_head.py [3]
- Ono sto je bitno navesti da ova klasa nasledjuje drugu klasu istog imena iz druge biblioteke gdje je u stvari implementiran *Mask2Former*, putanja do te biblioteke i pomenute klase: https://github.com/open-mmlab/mmdetection/blob/3.x/mmdet/models/dense_heads/mask2former_head.py [4]
- Pa se moja analiza u nastavku bavi samo tom bibliotekom.

Pixel Decoder

- U opisu arhitekture je navedeno da poslije *backbone* dolazi *pixel decoder.*

Pixel decoder. Mask2Former is compatible with any existing pixel decoder module. In MaskFormer [14], FPN [33] is chosen as the default for its simplicity. Since our goal is to demonstrate strong performance across different segmentation tasks, we use the more advanced multi-scale deformable attention Transformer (MSDeformAttn) [66] as our default pixel decoder. Specifically, we use 6 MSDeformAttn layers applied to feature maps with resolution 1/8, 1/16 and 1/32, and use a simple upsampling layer with lateral connection on the final 1/8 feature map to generate the feature map of resolution 1/4 as the per-pixel embedding. In our ablation study, we show that this pixel decoder provides best results across different segmentation tasks.

Slika 2. Opis *Pixel Decodera*

- Vidimo da je u opisu navedeno da se koristi *MSDeformAttn* i kod to dokazuje.
- U [1] na linijama 39-66 (pixel_decoder=dict(type='mmdet.MSDeformAttnPixelDecoder')) je navedeno da se koristi klasa *MSDeformAttnPixelDecoder* na putanji https://github.com/open-mmlab/mmdetection/blob/3.x/mmdet/models/layers/msdeformattn_pixel_decoder.py [5]
- Definisanje klase:
    - U [5] na linijama 46-47 je definisano da se koriste izlazi iz *backbone* sa dimenzijama [256, 512, 1024, 2048] i to je razlika u odnosu na rad, jer je u radu receno da se samo poslednje 3 dimenzije pustaju kroz enkoder, a prva se samo *upsampluje* da bi bila iste dimenzije. U kodu se prolazi kroz sve dimenzije jednako i isto se postupa sa njima, to se moze vidjeti na liniji 164 u [5].
    - Na linijama 67-78 definise se konvolucija za svaku dimenziju, koja proizvodi iste dimenzije jer je *kernel_size=1*
    - Na liniji 80 se definise enkoder *Mask2FormerTransformerEncoder* na putanji https://github.com/open-mmlab/mmdetection/blob/3.x/mmdet/models/layers/transformer/mask2former_layers.py [6] o kojem ce biti vise rijeci kasnije
    - Na liniji 81 definise se pozicioni enkoding koji koristi sinusoide
    - Na liniji 82 definise se level enkoding koji ce isto definisati pozicioni enkoding
- *Forward pass:*
    - Na linijama 144-246 (def forward(self, feats: List[Tensor]) -> Tuple[Tensor, Tensor]:)
    - Prima *feature* matricu od *backbone*
    - Proizvodi kako je definisano u komentaru:

```
Returns:
    tuple: A tuple containing the following:

        - mask_feature (Tensor): shape (batch_size, c, h, w).
        - multi_scale_features (list[Tensor]): Multi scale \
                features, each in shape (batch_size, c, h, w).
    """
```

- Na 164 liniji pocinje for petlja koja radi 4 puta, kako je definisano u konfigu i ona podesava ulaz za svaki *Mask2FormerTransformerEncoder* koji je vezan za po jednu dimenziju iz *backbone*
- Na linijama 165-168 proizvodi se *feature* matrica koja se dobija konvolucijom ulaza ali dimenzije ostaju iste, i ovo ce predstavljati *query*
- Na linijama 171-175 proizvodi se prazna matrica iste dimenzije kao ulaz, dodaju se pozicioni i level enkodinzi koji se uce i ovo ce predstavljati *key*
- Na linijama 177-178 se proizvode referentne tacke za enkoder
- Ostatak for petlje prilagodjava dimenzije ulazu u enkoder
- Na linijama 214-223 se poziva enkoder
- Na linijama 232-243 se primjenjuje bilinearna interpolacija i formiraju se povratne vrijednosti

- *Mask2FormerTransformerEncoder* na putanji [6]:
  - Na linijama 10-53 (class Mask2FormerTransformerEncoder(DeformableDetrTransformerEncoder):)
  - Nasledjuje klasu *DeformableDetrTransformerEncoder* koji je na putanji https://github.com/open-mmlab/mmdetection/blob/3.x/mmdet/models/layers/transformer/deformable_detr_layers.py [7]
  - Samo primjenjuje forward pass iz *DeformableDetrTransformerEncoder*
- *DeformableDetrTransformerEncoder* na putanji [7]
  - Na linijama 21-80 (class DeformableDetrTransformerEncoder(DetrTransformerEncoder):)
  - Racuna referentne tacke na linijama 82-118
  - Na linijama 70-80 samo poziva slojeve koji su *DeformableDetrTransformerEncoderLayer* i definisani su na 237-249(class DeformableDetrTransformerEncoderLayer(DetrTransformerEncoderLayer):), a primjenjuju *self attention* koji je *MultiScaleDeformableAttention* i uvezen je iz druge biblioteke (*from mmcv.ops import MultiScaleDeformableAttention*), zatim *feed forward network* i normalizaciju.
- Na kraju izlaz iz *pixel decodera* jesu maske i *feature matrice* koje ce ici u *Transformer Decoder.*

Transformer Decoder

- U definiciji *Mask2Former Head* na putanji [4] na linijama 103-104 je definisano da se kao *Transformer Decoder* koristi klasa *DetrTransformerDecoder* koja se nalazi na putanji

https://github.com/open-mmlab/mmdetection/blob/3.x/mmdet/models/layers/transformer/detr_layers.py [8] i klasa je definisana na linijama 84-159 (class DetrTransformerDecoder(BaseModule):)

- Ovo je razlika u odnosu na naucni rad jer su oni naveli da se kao *Transformer Decoder* koristi dekoder kao i u *MaskFormeru* ali ovdje se koristi *DETR* dekoder. Sto znaci da se *DETR* enkoder koristi u *Pixel Decoder* dijelu, a *Detr* dekoder u *Transformer Dekoder* dijelu.
- Navedena klasa samo definise svoje slojeve na linijama 112-120 a ti slojevi su *DetrTransformerDecoderLayer* i nalaze se u istom fajlu na linijama 241-374 (class DetrTransformerDecoderLayer(BaseModule):) i poziva *forward pass* za svaki sloj.
- *DetrTransformerDecoderLayer* klasa kao forward prolaz u linijama 312-374 radi klasican prolaz transformera, ide prvo *self attention* sloj pa normalizacija, pa *cross attention* pa normalizacija pa *feed forward network* pa normalizacija. To je isto razlika u odnosu na rad jer su oni naveli da su zamjenili mjesta *self attention* i *cross attention* sloja.
- U istoj klasi na linijama 300-310 je uradjena inicijalizacija slojeva, *self attention* i *cross attention* su implementirani kao obican *MultiheadAttention* koji je na liniji 6 uvezen iz druge biblioteke (from mmcv.cnn.bricks.transformer import FFN, MultiheadAttention) zajedno sa *FFN* slojem. Ovo je isto razlika u odnosu na rad jer su tamo naveli da su *cross attention* sloj zamjenili sa novim *mask attention* slojem a ja to nisam pronasao u kodu.

Implementacija Decode Head

- Klasa *MaskFormerHead* koja se nalazi u https://github.com/open-mmlab/mmdetection/blob/3.x/mmdet/models/dense_heads/mask2former_head.py [4] pocinje na liniji 23 (class MaskFormerHead(AnchorFreeHead):) prima podatke dobijene iz *Backbone* i poziva gore navedene *Pixel Decoder* i *Transformer Decoder* i racuna *loss.*
- Na linijama 23-123 radi se inicijalizacija svih potrebnih dijelova.
- Glavna funkcija koja se poziva jeste na liniji 524-564 i naziva se *loss* (def loss(self,x: Tuple[Tensor],batch_data_samples: SampleList,) -> Dict[str, Tensor]:)
- Prima dobijene matrice koje su procesirane od strane *Backbone.*
- Takodje dobija parametar koji definise da li se radi semanticka, panopticka ili instance segmentacija.
- Zatim poziva forward funkciju koja je implementirana na linijama 455-522(def forward(self, x: Tuple[Tensor],batch_data_samples: SampleList) -> Tuple[Tensor]:).
- Ova funkcija prilagodjava ulaz *Pixel Decoderu,* zatim ga poziva na liniji 492, zatim racuna pozicioni *embedding* na liniji 492 koji je implementiran kao (self.decoder_pe = SinePositionalEncoding(**positional_encoding)), zatim na linijama 506-512 poziva *Transformer Decoder* gdje su:
    - *Query* vrijednosti definisani na linijama 503-504 inicijalizovani 0-ma.
    - *Key* i *Value* postavljeni na izlaz iz *Pixel Decoder*
    - I prethodno navedeni pozicioni embedinzi su poslani da se mogu uciti

- Dalje u istoj funkciji definisano je racunanje predvidjenih klasa koje se dobiju tako sto se propuste kroz jedan Linearan sloj, definicija na 116 (self.cls_embed = nn.Linear(feat_channels, self.num_classes + 1)), i racunanje maski koje je definisano na linijama 117-120. To racunanje je uradjeno na linijama 515-520.
- Maske su dobijene *xor* operacijom nad rezultatom *Pixel Decoder* i rezultatom *Transformer Decoder.*
- Na kraju ova funkcija vraca predvidjene maske i klase.
- Nazad u loss funkciji na liniji 557 se poziva funkcija *preprocess_gt* definisana na 144-190 (def preprocess_gt(self, batch_gt_instances: InstanceList,batch_gt_semantic_segs: List[Optional[PixelData]]) -> InstanceList:) koja prima ulazne matrice i racuna prave maske i prave klase.
- Dalje u loss funkciji na liniji 561 se poziva funkcija *loss_by_feat* definisana na linijama 324-365 (def loss_by_feat(self, all_cls_scores: Tensor, all_mask_preds: Tensor,batch_gt_instances: List[InstanceData],batch_img_metas: List[dict]) -> Dict[str, Tensor]:), u koju se salju prave maske i klase, i predvidjene klase i maske.
- Ova funkcija poredi ove dvije stvari, i racuna 3 losa:
    - *loss_cls (*definisan kao loss_cls: ConfigType = dict(type='CrossEntropyLoss',use_sigmoid=False,loss_weight=1.0,class_weight=[1.0] * 133 + [0.1]),) koji racuna gresku predvidjene klase,
    - *loss_mask* (definisan kao loss_mask: ConfigType = dict(type='FocalLoss',use_sigmoid=True,gamma=2.0,alpha=0.25,loss_weight=20.0),) koji racuna gresku predvidjene maske
    - *loss_dice* (definisan kao loss_dice: ConfigType = dict(type='DiceLoss',use_sigmoid=True,activate=True,naive_dice=True,loss_weight=1.0),)
- Ova 3 losa se zapakuju u *dict* objekat koji se vraca.

Reference

- MMsegmentation github repository version v1.2.0 (downloaded 23.1.2024.) - https://github.com/open-mmlab/mmsegmentation/tree/main
- MMdetection github repository version v.3.3.0. (downloaded 23.1.2024.) - https://github.com/open-mmlab/mmdetection/tree/3.x
- MaskFormer naucni rad - https://arxiv.org/pdf/2107.06278v2.pdf
- Mask2Former naucni rad - https://arxiv.org/pdf/2112.01527v3.pdf
- Deformable DETR: Deformable Transformers for End-to-End Object Detection - https://arxiv.org/abs/2010.04159

Uputstva za implementaciju

- Prvi problem jeste da nije dovoljno ispratiti upustva koja su navedena u zvanicnoj dokumentaciji za implementaciju Mask2Formera. Pored instalacije mmsegmentation paketa potrebno je instalirati i mmdetection paket i to na ovaj nacin:

```
!pip install openmim
!mim install 'mmcv >= 2.0.0rc1'

# Install mmseg
!git clone -b main https://github.com/open-mmlab/mmsegmentation.git
!cd mmsegmentation && pip install -e .

# Install mmdet
!git clone https://github.com/open-mmlab/mmdetection.git
!cd mmdetection && pip install -e .
```

- Isto tako u dokumentaciji nije navedeno da je potrebno instalirati paket ftfy koji je neophodan.

```
!pip install ftfy
```

- Slike iz baze ADE20K nisu istih dimenzija tako da je potrebno sve ih pretvoriti u dimenzije 512x512.
- Potrebno je pronaci paletu boja za klase iz baze ADE20K.
- Naredna linija koda skida odgovarajuci model i njegovu konfiguraciju da bi se mogao iskoristiti:

```
# Download config and checkpoint files
!mim download mmsegmentation --config mask2former_r50_8xb2-160k_ade20k-512x512 --dest .
```

- Podesavanje config fajla:
  - Ucitati skinuti config fajl iz prethodnog koraka kao i njegove tezine

```
cfg = Config.fromfile('configs/mask2former/mask2former_r50_8xb2-160k_ade20k-512x512.py')

# Load the pretrained weights
cfg.load_from = 'mask2former_r50_8xb2-160k_ade20k-512x512_20221204_000055-2d1f55f1.pth'
```

  - Definisati dimenziju slika i broj klasa modela

```
cfg.crop_size = (512, 512)  # Change this: desired crop
cfg.model.data_preprocessor.size = cfg.crop_size

# Change this: set number of classes
cfg.model.decode_head.num_classes = 150
cfg.num_classes = 150
```

- Postaviti putanju do klase koja ce raditi ucitavanje podataka u model kao i putanju do foldera koji sadrzi slike za treniranje

```
cfg.dataset_type = 'Ade20KDataset'  # Name of the dataset you want to use
cfg.data_root = data_root # Directory in which you have images/ and labels/ folders
```

- Definisati train i test pipeline, koji ucitava slike i anotacije. (Napomena: ovdje se moze ubaciti mijenjanje dimenzija slika da bi se to automatski uradilo)

```
cfg.train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations'),
    dict(type='PackSegInputs')
]

cfg.test_pipeline = [
    dict(type='LoadImageFromFile'),
    # add loading annotation after ``Resize`` because ground truth
    # does not need to do resize data transform
    dict(type='LoadAnnotations'),
    dict(type='PackSegInputs')
]
```

- Postaviti broj iteracija treninga i validacije

```
cfg.train_cfg.max_iters = 100
cfg.train_cfg.val_interval = 100
cfg.default_hooks.logger.interval = 5
cfg.default_hooks.checkpoint.interval = 100
```

- Ucitati tezine klasa

```
weights = [1.0] * cfg.num_classes
weights.append(0.1) # Expected if you look at original config, need 0.1 as last item
cfg.model.decode_head.loss_cls["class_weight"] = weights
```

- Pokrenuti treniranje

```
runner = Runner.from_cfg(cfg)
runner.train()
```

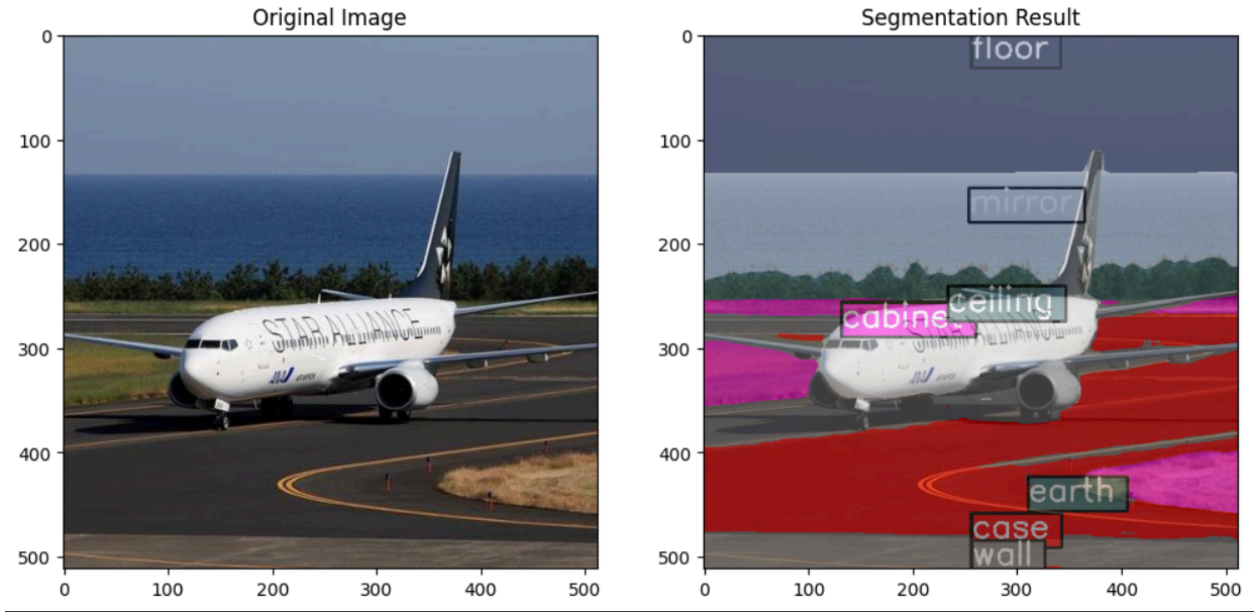
- Optimizator je ostavljen kao predefinisani:

```
# optimizer
embed_multi = dict(lr_mult=1.0, decay_mult=0.0)
optimizer = dict(
    type='AdamW', lr=0.0001, weight_decay=0.05, eps=1e-8, betas=(0.9, 0.999))
```

- Naredni dio koda radi inference treniranog modela:

```
# Init the model from the config and the checkpoint
checkpoint_path = './work_dirs/tutorial/iter_100.pth'
model = init_model(cfg, checkpoint_path, 'cuda:0')

# Load the original image
img = mmcv.imread('ade20k/images/ADE_train_00014559.jpg')

# Perform inference to get segmentation result
result = inference_model(model, img)

# Create a subplot with 1 row and 2 columns
plt.figure(figsize=(12, 6))

# Plot the original image in the first subplot
plt.subplot(1, 2, 1)
plt.imshow(mmcv.bgr2rgb(img))
plt.title('Original Image')

# Plot the segmentation result in the second subplot
plt.subplot(1, 2, 2)
vis_result = show_result_pyplot(model, img, result)
plt.imshow(mmcv.bgr2rgb(vis_result))
plt.title('Segmentation Result')

# Show the subplots
plt.show()
```

-
- Dobijeni rezultati izgleda ovako:


Original Image
Segmentation Result
floor
mirror
ceiling
cabinet
earth
case
wall
0
100
200
300
400
500