# Note méthodologique – Preuve de concept

**Sujet :** Segmentation sémantique d’images de scènes urbaines  
**Baseline :** VGG16‑UNet (TensorFlow/Keras)  
**Modèle récent :** SegFormer (MiT‑B0/B2, TensorFlow/Keras)

## 1) Dataset retenu

**Dataset : Cityscapes** (segmentation sémantique d’environnements urbains).

* **Contenu :** images haute résolution (≈ 2048×1024) capturées dans **50 villes** ; **5 000** images **finement annotées** (fine) et **20 000** images **grossièrement annotées** (coarse).
* **Tâche :** segmentation sémantique en **19 classes** (route, trottoir, bâtiment, piéton, voiture, etc.).
* **Splits usuels :** train ≈ 2 975, val ≈ 500, test ≈ 1 525 (test sans labels publics).
* **Pourquoi ce dataset ?** Référence académique pour la conduite autonome (variabilité des scènes, forte densité d’objets et de frontières).

**Pré‑traitement & préparation :** - Conversion des masques vers l’espace de classes à 8 labels, **ignore\_index** pour les pixels hors taxonomie.  
- Normalisation ImageNet ; **resize/crop** aléatoires ; **flip** horizontal ; **color jitter** léger.  
- Génération d’un **DataLoader**/tf.data performant (cache, prefetch, mixed precision si GPU compatible).

## 2) Les concepts du modèle récent (SegFormer)

**SegFormer (2021)** = **MiT (Mix Transformer) encodeur hiérarchique** + **décodeur tout‑MLP** très léger :

* **Encodeur MiT** : pyramide multi‑échelles ; **patch embeddings qui se recouvrent** ; attention locale/globale ; **pas de positional encoding** ⇒ meilleure **robustesse au changement de résolution**.
* **Décodeur MLP** : agrégation simple des features de plusieurs niveaux (sans ASPP, sans FPN complets) ⇒ **latence faible** et **faible nombre de paramètres**.
* **Famille de tailles** B0→B5 (compromis précision/vitesse).
* **Atouts clés** : bonnes performances mIoU sur Cityscapes et ADE20K ; pipeline simple ; efficacité en inférence.

**Différences vs U‑Net/VGG16** : - U‑Net repose sur **skip connections** convolutionnelles symétriques ; SegFormer exploite un **encodeur Transformer hiérarchique** et un **décodeur minimaliste**.  
- SegFormer gère naturellement les **contexte longue portée** ; U‑Net est très fort en **frontières fines** et en données limitées.

## 3) Modélisation

### 3.1. Protocole commun

* **Entrées** : images RGB ; tailles d’entraînement typiques **512×512** (crops) ou **768×768** selon VRAM.
* **Loss** : SparseCategoricalCrossentropy(from\_logits=True) + **pondération de classes** si déséquilibre ; variante **Focal** ou **Dice** si nécessaire.
* **Optimisation** : **AdamW** (lr de base 3e‑4, weight\_decay 1e‑4) ; **warmup** 5–10 epochs puis **cosine decay**.
* **Régularisation** : data augmentation, **label smoothing** (p. ex. 0.05), **dropout** léger sur la tête.
* **Arrêt** : early stopping sur **mIoU val** ; checkpoint meilleur modèle.

### 3.2. Baseline – VGG16‑UNet (Keras)

* **Backbone** : VGG16(weights="imagenet", include\_top=False) ; geler partiellement (stages bas) puis **unfreeze progressif**.
* **Decoder** : chemins up‑convs + **skip connections** (U‑Net classique) ; dernière conv 1×1 vers 19 sorties (logits).
* **Hyperparams** : batch 8–16 ; 80–120 epochs ; input 512 ; scheduler cosine ; mix‑precision.

### 3.3. Modèle récent – SegFormer (TensorFlow/Keras)

* **Implémentation** : Keras/TF (MiT‑B0 ou B2) ; tête SegFormer (décodeur MLP).
* **Initialisation** : poids pré‑entraînés ImageNet‑1k (MiT) si disponibles ; sinon initialization Kaiming/LeCun.
* **Hyperparams** : identiques autant que possible à la baseline ; variantes testées : taille d’entrée, batch, lr.
* **Détails d’entraînement** :
  + optimizer = AdamW(3e‑4, weight\_decay=1e‑4) ; **warmup** linéaire 1–5k iters ; **cosine** schedule.
  + **Ablations** : sans/avec TTA au test ; crops 512 vs 768 ; loss CE vs CE+Dice.

### 3.4. Métriques & évaluation

* **Principale** : **mIoU** (macro), calculée sur 19 classes.
* **Secondaires** : **IoU par classe**, **pixel accuracy**, **latence** (ms/image), **taille du modèle** (Mo).

### 3.5. Optimisation

* **Recherche d’hyperparamètres** légère (grid/random) sur lr ∈ {1e‑3, 3e‑4, 1e‑4}, batch ∈ {8, 16}, taille ∈ {512, 768}.
* **Choix final** par **mIoU val** ; tester la **stabilité** via 3 seeds.

## 4) Synthèse des résultats (exemple de présentation)

| Modèle | mIoU (val) | Pixel Acc. | Latence (ms, 512²) | Paramètres (M) |
| --- | --- | --- | --- | --- |
| **VGG16‑UNet (baseline)** | **…** | … | … | … |
| **SegFormer‑MiT‑B0** | **…** | … | … | … |
| **SegFormer‑MiT‑B2** *(optionnel)* | **…** | … | … | … |

* **Gains** : ΔmIoU = … points ; Δlatence = … ms ; Δtaille = … Mo.
* **Analyse qualitative** : overlays de masques (route/trottoir/véhicules/piétons) ; meilleure séparation bordures ? gestion du lointain ?

**À compléter après entraînement** : insérer chiffres, graphiques (barres IoU par classe, exemples d’images).

## 5) Interprétabilité – importance globale & locale

* **Globale** :
  + **IoU par classe** (bar chart) = importance « fonctionnelle » des classes ;
  + **Erreurs typiques** : confusions trottoir/route, piéton/poteau, végétation/bâtiment ;
  + **Sensibilité** aux tailles d’objets (petits piétons/feux) et aux **frontières**.
* **Locale** :
  + **U‑Net** : **Grad‑CAM / Grad‑CAM++** sur couches conv profondes ;
  + **SegFormer** : **attention rollout** (encodeur) + **occlusion sensitivity** (masquage de patchs) ;
  + **Explicabilité par instance** : superposer cartes de chaleur sur l’image, commenter décisions correctes/erreurs.

## 6) Limites & améliorations possibles

* **Données** : *domain shift* (nuit/pluie), classes rares ⇒ **re‑échantillonnage**, **loss pondérée** ou **Focal**.
* **Modèle** : essayer **MiT‑B2/B4**, **plus haute résolution** (768–1024), **TTA** au test, **multi‑scale training**.
* **Entraînement** : **scheduler** plus long (160k iters), **EMA** des poids, **MixUp/CutMix** légers.
* **Post‑traitement** : CRF/affinage de bordures si nécessaire.
* **Déploiement** : quantification 8‑bits / pruning pour latence embarquée.

## 7) Reproductibilité

* **Environnement** : Python 3.10+, TensorFlow/Keras 2.15+ ou Keras 3, CUDA 12.x, cuDNN 9.x.
* **Matériel** : GPU 16–24 Go (A5000/RTX 4090) recommandé pour 768² ; 8–12 Go pour 512².
* **Seeds** : fixer numpy, python, tf.random ; log MLflow/W&B ; sauvegarder splits.

## 8) Références (sélection)

* **SegFormer** – Xie et al., NeurIPS 2021.
* **Cityscapes** – Cordts et al., 2016 (site officiel).
* **U‑Net** – Ronneberger et al., 2015.
* **VGG16** – Simonyan & Zisserman, 2014.
* **mIoU** – définitions courantes (doc outillage métriques).

Les chiffres exacts et figures seront ajoutés après exécutions finales (mêmes splits et métriques pour baseline et SegFormer).