Développez une preuve de concept

ABBAS Billel

Octobre 2024

Agenda

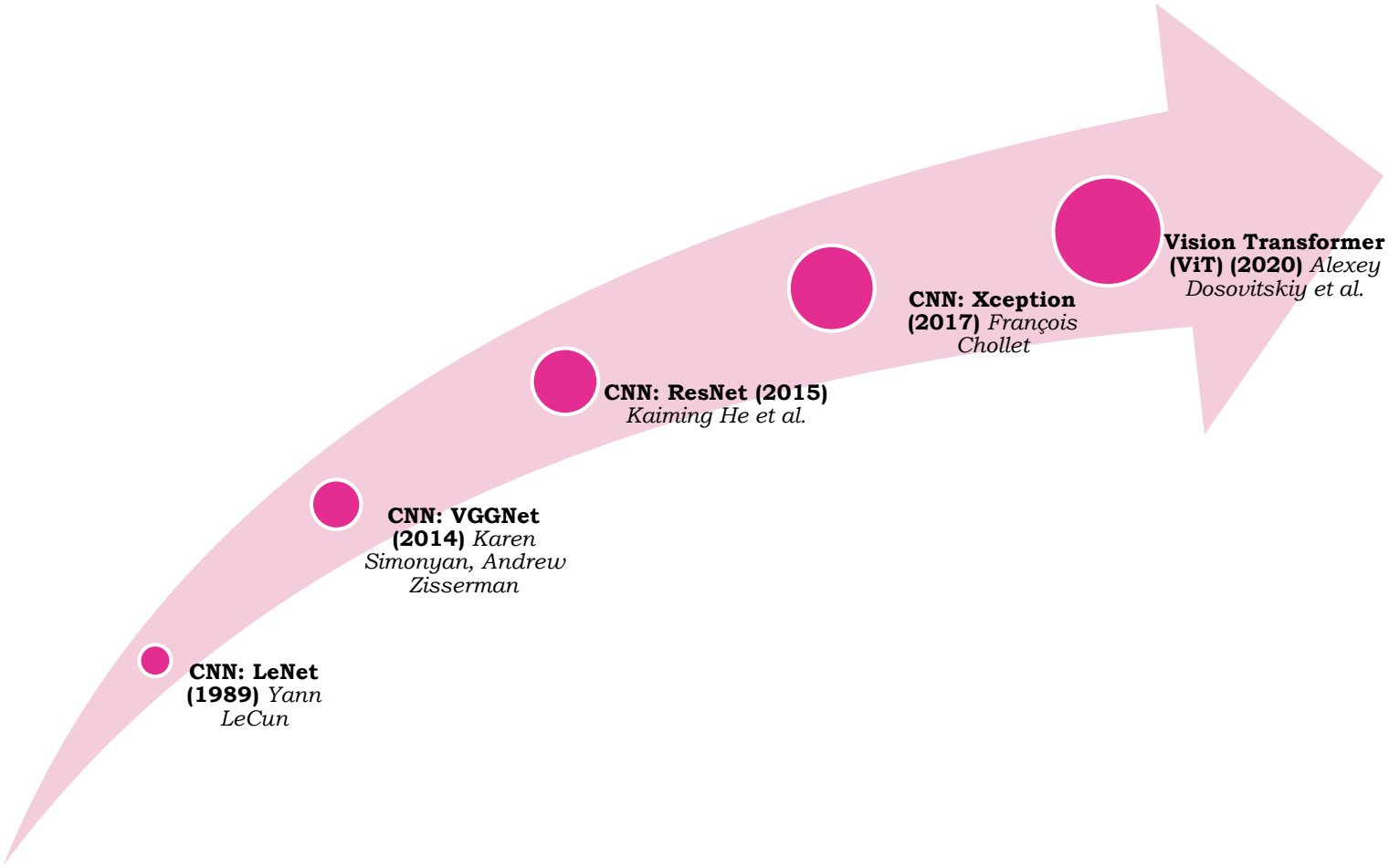
- **Introduction au projet**
- **Historique des avancées en classification d'images**
- **Axes d'amélioration du projet 6**
  - Extraction des données et choix des 5 races de chiens
  - Rappel du Principe de la convolution dans les CNN
  - Fine-tuning du modèle Xception
  - Résultats du modèle Xception
- **Modèle ViT (Vision Transformer)**
  - Présentation du modèle ViT & sel-attention
  - ViT from scratch: Présentation
  - ViT from scratch: Résultats
  - ViT en transfer learning: Présentation
  - ViT en transfer learning: Résultats
- **Sélection du meilleur modèle**
- **Comparaison entre modèle Xception (CNN) et ViT (Transformers)**
- **Benchmark des nouveaux modèles**
  - Swin Transformers : Présentation
- **Conclusion**

Introduction au projet

- **Contexte du projet :**
  - Dans le cadre du recrutement chez DataSpace, une entreprise spécialisée en **data science**, ce projet vise à comparer les performances des **modèles convolutionnels (CNN)**, en particulier le modèle **Xception** du projet 6, avec les **Vision Transformers (ViT)**, une nouvelle approche en vision par ordinateur pour traiter des images complexes.
- **Problématique :**
  - Le domaine du **machine learning** évolue rapidement, il est donc essentiel de se tenir informé des dernières avancées. Ce projet cherche à déterminer si les **transformers**, initialement développés pour le **traitement du langage**, peuvent surpasser les **CNN** dans la **vision par ordinateur** pour la classification d'images.
- **Approche :**
  - **Modèles CNN :** Utilisation du modèle **Xception** comme baseline, avec des améliorations comme le fine-tuning et l'early stopping.
  - **Vision Transformer (ViT) :** Comparaison entre un modèle **ViT** entraîné **from scratch** et un autre via **transfer learning**, qui utilise des mécanismes d'attention pour capturer des relations globales.
  - **Comparaison** des deux approches en termes de **précision**, **scalabilité** et **complexité**.
- **Objectif :**
  - Montrer les avantages des **Vision Transformers (ViT)** face aux **CNN** pour la classification d'images, tout en développant une **API** pour prédire la classe d'une image avec le modèle **ViT**.

Historique des avancées en classification d'images:

Évolution des modèles de classification d'images (1989-2020):

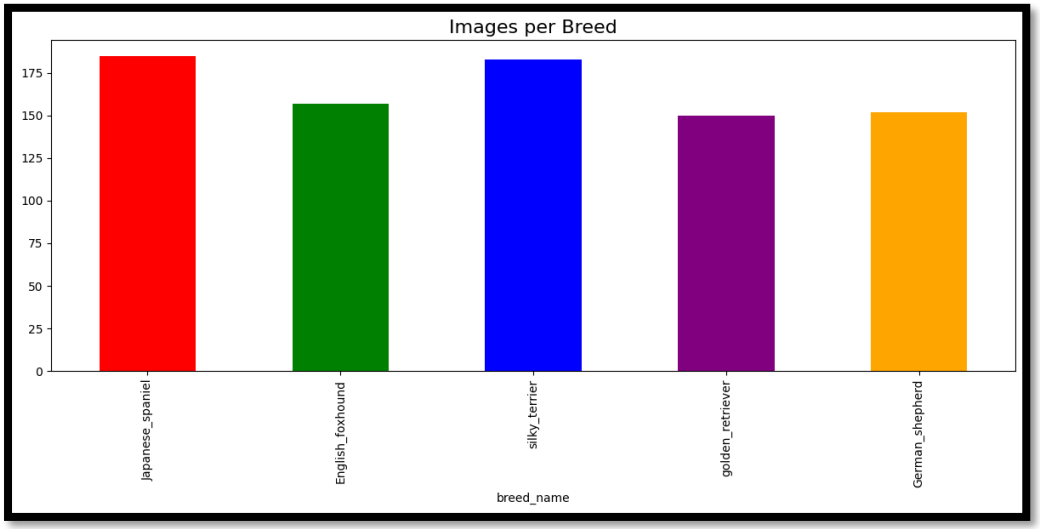


Axes d'amélioration du projet 6:

Extraction des données:

Lien vers le site universitaire de Stanford : [Stanford Dogs Dataset]

Extension de **la classification de 3** à **5 races** de chiens suite aux améliorations du **projet 6**

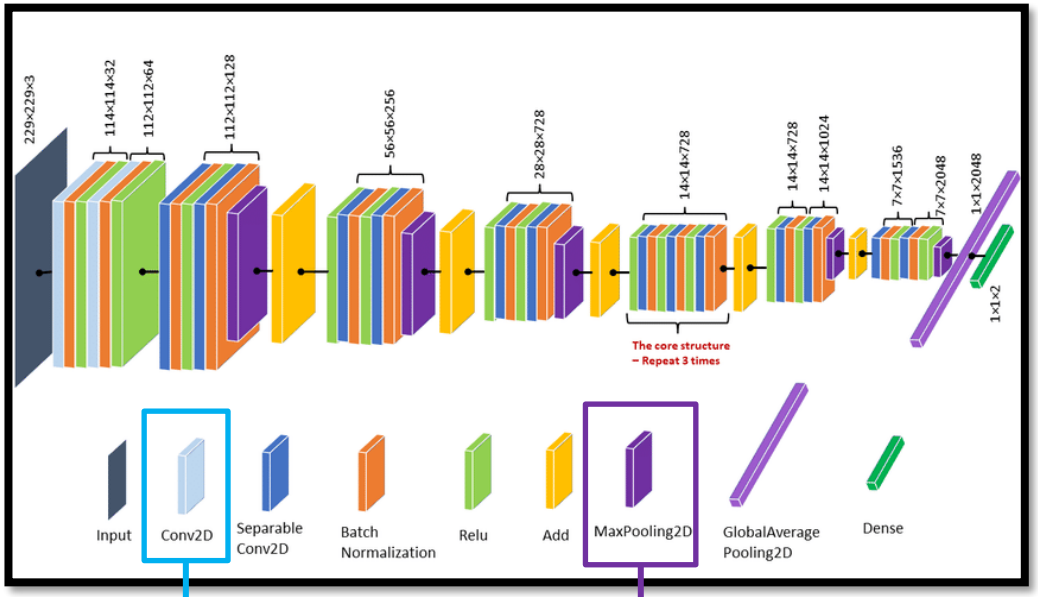


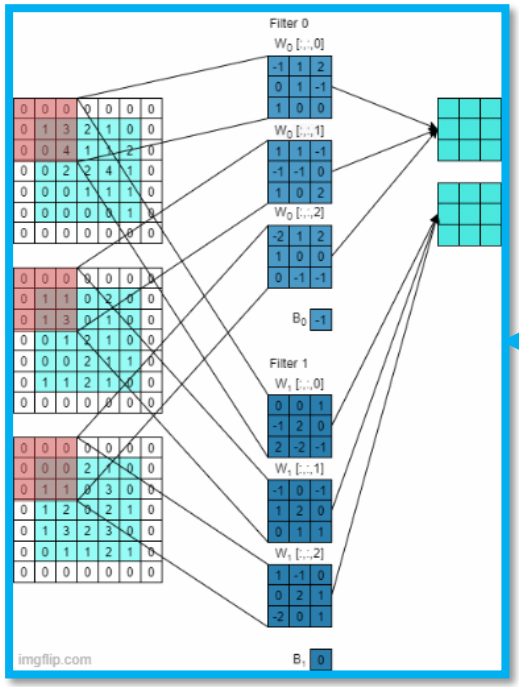
- **827 images**
- **5 Races**
- **German_shepherd 152 Photos**
- **Silky_terrier 183 Photos**
- **Golden_retriever 150 Photos**
- **Japanese Spaniel : 178 photos**
- **English Foxhound : 152 photos**

Axes d'amélioration du projet 6

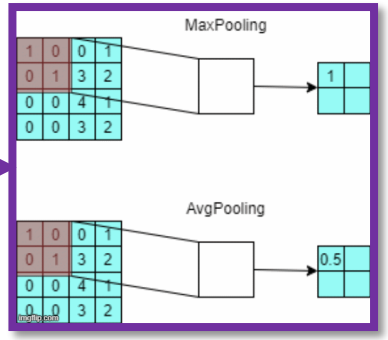
Rappel du Principe de la convolution dans les CNN

Architecture du modèle Xception :
Visualisation des différentes couches





Animation du **principe** de la **convolution** dans un **CNN**



Animation du **principe** de **MaxPooling** et **AvgPooling** dans un **CNN**

Axes d'amélioration du projet 6

Fine-tuning du modèle Xception

Adam: learning_rate = 5e-5

```python
def create_xception_model(input_shape=(299, 299, 3), num_classes=5, dropout_rate=0.3, fine_tune_start=51):
    # Charger le modèle Xception pré-entraîné sur ImageNet sans les couches fully connected
    base_model_xception = Xception(weights='imagenet', include_top=False, input_shape=input_shape)

    # Geler les premières couches du modèle Xception jusqu'à `fine_tune_start
    for layer in base_model_xception.layers[:fine_tune_start]:
        layer.trainable = False
    for layer in base_model_xception.layers[fine_tune_start:]:
        layer.trainable = True

    # Créer un modèle séquentiel
    model = Sequential([
        # Ajouter le modèle Xception pré-entraîné comme base
        base_model_xception,

        # Ajouter les nouvelles couches fully connected
        GlobalAveragePooling2D(),  # Pooling global pour réduire la dimensionnalité
        Dense(512),  # Couche Dense avec 512 unités
        BatchNormalization(),  # Normalisation pour stabiliser l'apprentissage
        Activation('relu'),  # Activation ReLU
        Dropout(dropout_rate),  # Dropout pour éviter le surapprentissage
        Dense(num_classes, activation='softmax')  # Couche de sortie pour la classification avec Softmax
    ])

    return model
```

Le **fine-tuning** consiste à **geler les premières couches** du modèle **Xception pré-entraîné sur ImageNet**, tout **en dégelant** et **réentraînant les couches supérieures** à partir de **la 51e couche**.

```python
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=20,
    restore_best_weights=True,
    min_delta=0.001,
    verbose=1
)
```

L'entraînement s'arrêtera si la **val_loss** ne s'améliore pas pendant 20 epochs, en restaurant les **meilleurs poids**. Il s'arrête aussi si l'amélioration est inférieure à **0.001.**

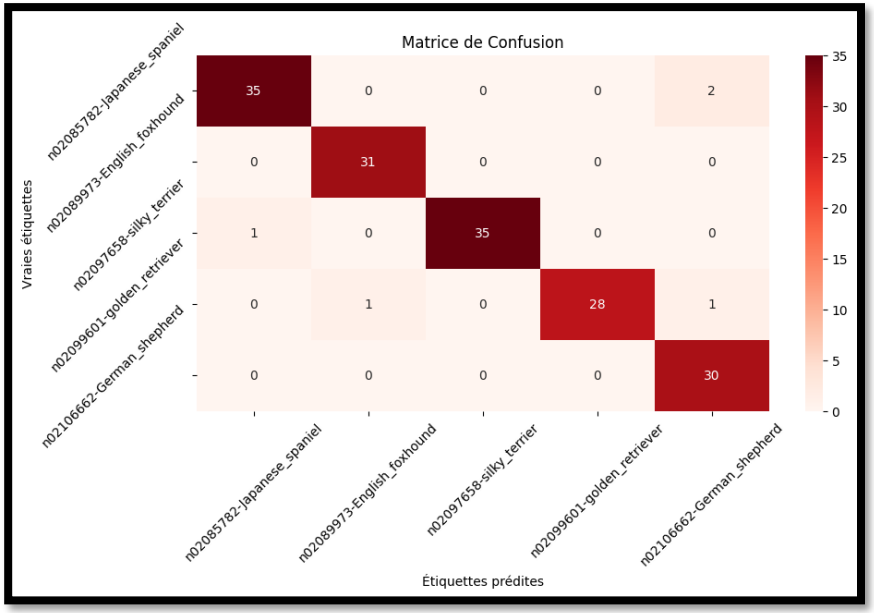
Les axes d'amélioration du projet 6 ont été appliqués : le nombre de classes est passé de 3 à **5 races** de chiens, **l'early stopping** a été **mis en place**, le **learning rate** est passé de 1e-4 (projet 6) à **5e-5** et le **fine-tuning** a été réalisé **en gelant les premières couches** et en **dégelant** puis **réentraînant** les couches supérieures **à partir de la couche 51**.

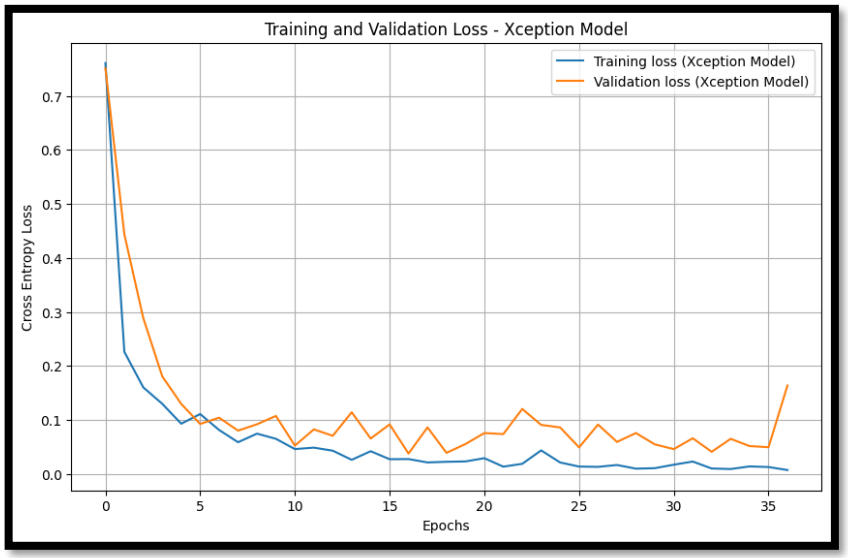
Axes d'amélioration du projet 6

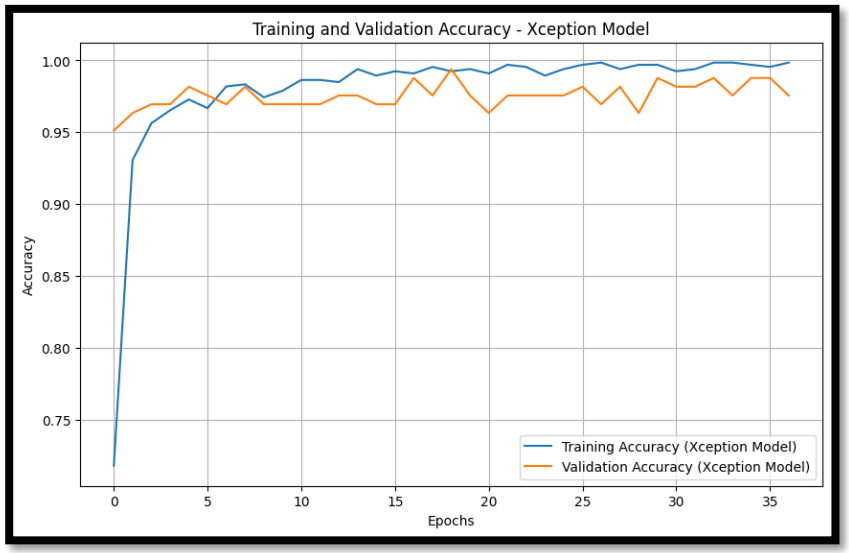
Résultats du modèle Xception

Résultats:

	precision	recall	f1-score	support
n02085782-Japanese_spaniel	0.97	0.95	0.96	37.00
n02089973-English_foxhound	0.97	1.00	0.98	31.00
n02097658-silky_terrier	1.00	0.97	0.99	36.00
n02099601-golden_retriever	1.00	0.93	0.97	30.00
n02106662-German_shepherd	0.91	1.00	0.95	30.00
accuracy	0.97	0.97	0.97	0.97
macro avg	0.97	0.97	0.97	164.00
weighted avg	0.97	0.97	0.97	164.00



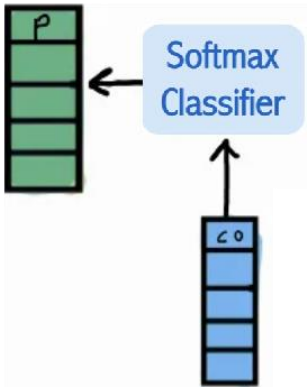




Modèle ViT (Vision Transformer)

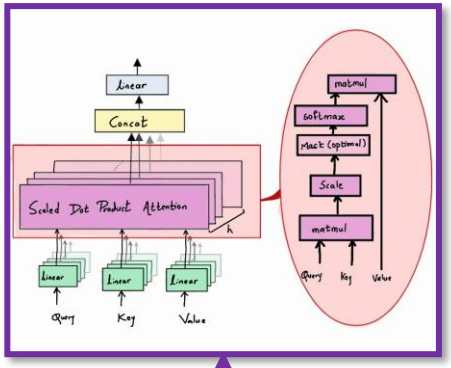
Présentation du modèle ViT & sel-attention

Probabilité de chaque classe



Multi-Head Attention Block

ATTENTION IS ALL YOU NEED



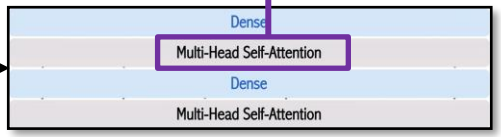
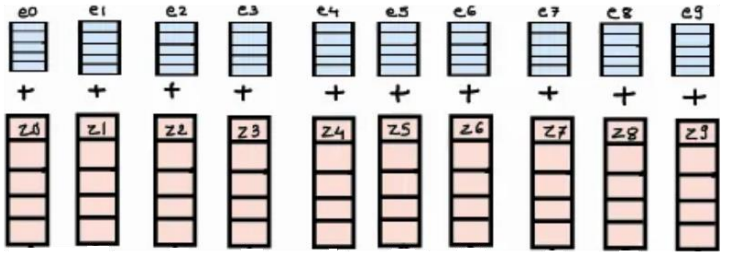
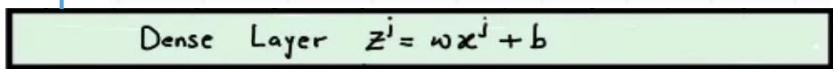


Image: 48 * 48

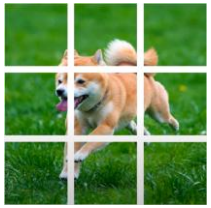
Encodage de la position

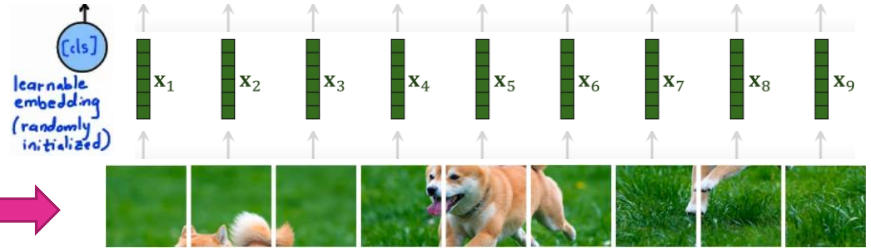


Linear patching



9 patches: 16 * 16





Reshape de chaque patch en un vecteur

Modèle ViT (Vision Transformer)

ViT from scratch: Présentation

Fonction PatchEncoder

```python
# Classe PatchEncoder
class PatchEncoder(layers.Layer):
    def __init__(self, num_patches, projection_dim):
        super().__init__()
        self.num_patches = num_patches
        self.projection = layers.Dense(units=projection_dim)
        self.position_embedding = layers.Embedding(
            input_dim=num_patches, output_dim=projection_dim
        )

    def call(self, patch):
        positions = tf.expand_dims(
            tf.range(start=0, limit=self.num_patches, delta=1), axis=0
        )
        projected_patches = self.projection(patch)
        encoded = projected_patches + self.position_embedding(positions)
        return encoded

    def get_config(self):
        config = super().get_config()
        config.update({"num_patches": self.num_patches})
        return config

# Fonction MLP (perceptron multicouche)
def mlp(x, hidden_units, dropout_rate):
    for units in hidden_units:
        x = layers.Dense(units, activation=keras.activations.gelu)(x)
        x = layers.Dropout(dropout_rate)(x)
    return x
```

Adam: learning_rate = 1e-5

La fonction PatchEncode dans ViT va:

- Projeter les **patches** en **vecteurs**
- Ajoute un **encodage de position** pour capturer leur ordre dans le **Vision Transformer**

Fonction MLP

```python
# Fonction MLP (perceptron multicouche)
def mlp(x, hidden_units, dropout_rate):
    for units in hidden_units:
        x = layers.Dense(units, activation=keras.activations.gelu)(x)
        x = layers.Dropout(dropout_rate)(x)
    return x
```

La fonction MLP dans ViT transforme les représentations des patches avec des couches non linéaires via **GELU**.

Modèle VIT

```python
# Modèle complet ViT (Vision Transformer)
def build_vit_model(input_shape=(224, 224, 3), num_classes=5, patch_size=16, num_patches=196,
                    projection_dim=64, transformer_layers=8, num_heads=4, transformer_units=[256, 64],
                    mlp_head_units=[128, 64]):

    inputs = keras.Input(shape=input_shape)
    patches = Patches(patch_size)(inputs)
    encoded_patches = PatchEncoder(num_patches, projection_dim)(patches)

    for _ in range(transformer_layers):
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
        attention_output = layers.MultiHeadAttention(num_heads=num_heads, key_dim=projection_dim, dropout=0.1)(x1, x1)
        x2 = layers.Add()([attention_output, encoded_patches])
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
        x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)
        encoded_patches = layers.Add()([x3, x2])

    representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
    representation = layers.Flatten()(representation)
    representation = layers.Dropout(0.5)(representation)
    features = mlp(representation, hidden_units=mlp_head_units, dropout_rate=0.5)
    logits = layers.Dense(num_classes, activation="softmax")(features)

    model = keras.Model(inputs=inputs, outputs=logits)
    return model
```

La fonction **ViT** suit ces étapes :

•**Divise l'image en patches** pour la transformer en séquences de vecteurs.

•**Applique des couches transformer**, intégrant **l'attention multi-têtes** et un **MLP**, pour apprendre les relations entre les patches.

•**Utilise un MLP final** pour générer les prédictions basées sur les caractéristiques visuelles extraites des patche

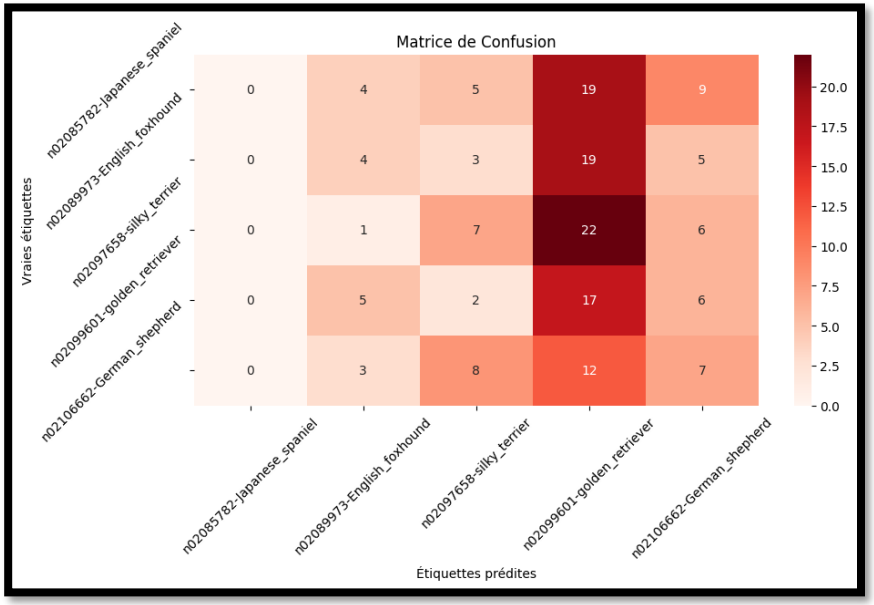
Code extrait du site: Image classification with Vision Transformer (keras.io)

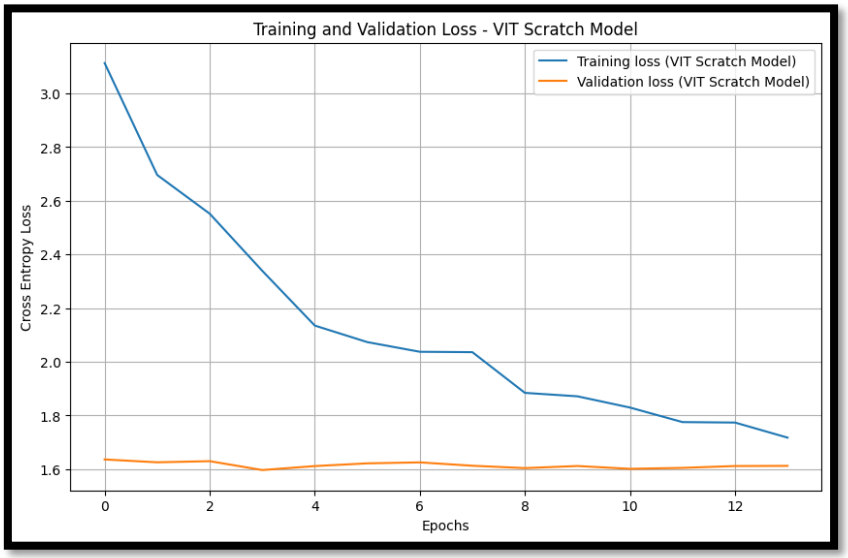
Modèle ViT (Vision Transformer)

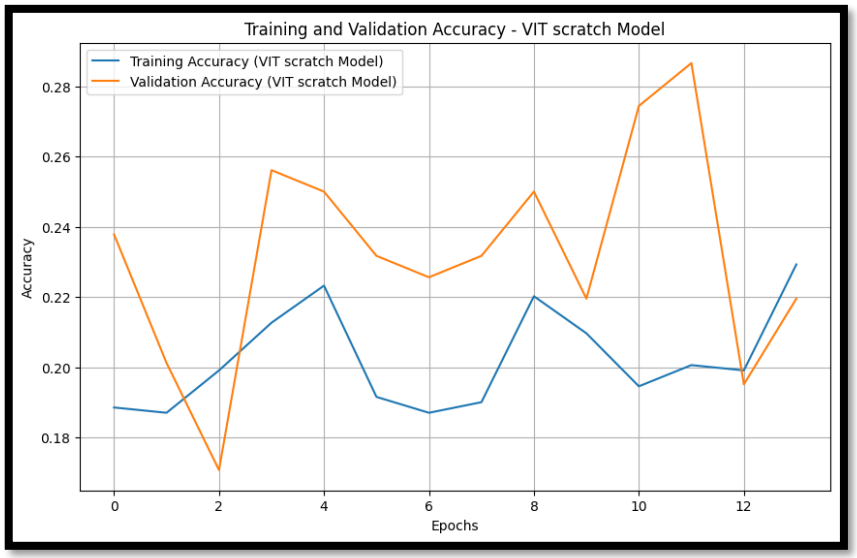
ViT from scratch: Résultats

Résultats:

	precision	recall	f1-score	support
n02085782-Japanese_spaniel	0.00	0.00	0.00	37.00
n02089973-English_foxhound	0.24	0.13	0.17	31.00
n02097658-silky_terrier	0.28	0.19	0.23	36.00
n02099601-golden_retriever	0.19	0.57	0.29	30.00
n02106662-German_shepherd	0.21	0.23	0.22	30.00
accuracy	0.21	0.21	0.21	0.21
macro avg	0.18	0.22	0.18	164.00
weighted avg	0.18	0.21	0.17	164.00







Modèle ViT (Vision Transformer)

ViT en transfer learning: Présentation

Adam: learning_rate = 1e-5

```python
# Modèle Vision Transformer (ViT)
def build_vit_transfer_model(input_shape=(224, 224, 3), num_classes=5):
    # Définir l'entrée du modèle avec la forme spécifiée
    inputs = Input(shape=input_shape)

    # URL du modèle Vision Transformer pré-entraîné sur TensorFlow Hub
    vit_model_url = "https://tfhub.dev/sayakpaul/vit_b16_fe/1"

    # Charger le ViT comme une couche Keras, marqué comme entraînable pour permettre le fine-tuning
    vit_layer = hub.KerasLayer(vit_model_url, trainable=True)

    # Passer les données d'entrée à travers le modèle ViT
    x = vit_layer(inputs)

    # Ajouter une couche Dense pour l'apprentissage des caractéristiques de haut niveau
    # Activation ReLU est utilisée pour ajouter de la non-linéarité
    x = Dense(512, activation='relu')(x)

    # Appliquer le dropout pour réduire le surajustement lors du fine-tuning
    x = Dropout(0.5)(x)

    # Couche de sortie avec activation softmax pour la classification
    # Softmax est utilisé pour calculer la probabilité de chaque classe
    outputs = Dense(num_classes, activation='softmax')(x)

    # Créer et retourner le modèle Keras
    model = Model(inputs=inputs, outputs=outputs)
    return model
```

1 Bloc de:

❖ **Base model ViT :**

- **Dé -Geler** les **couches du** modèle
- Input Shape = **(224,224,3)**
- La version du modèle **ViT** utilisée est **b_16**, correspondant à la version **'base'** avec des **patches** de **16 x 16**.
- Importer depuis : Vision-transformer_Tensorflow_hub

1 Bloc de:

❖ **Couche Dense**:

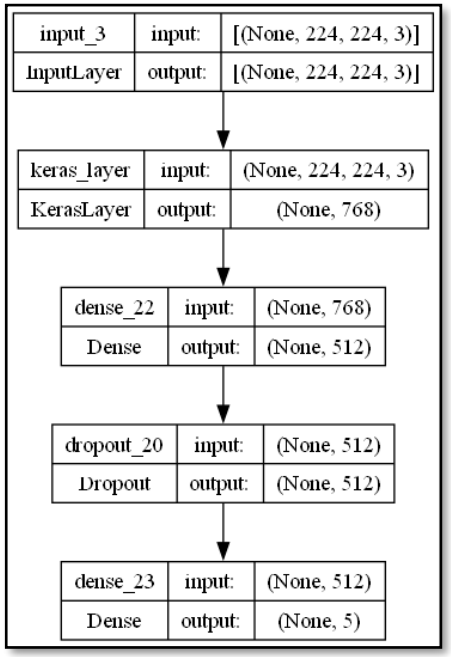
- **512** neurones

❖ Fonction d'activations = **'relu'**

❖ **Dropout (0.5)**

❖ **Couche Dense**:

- **5** neurones
- Fonction d'activations = **'softmax'**

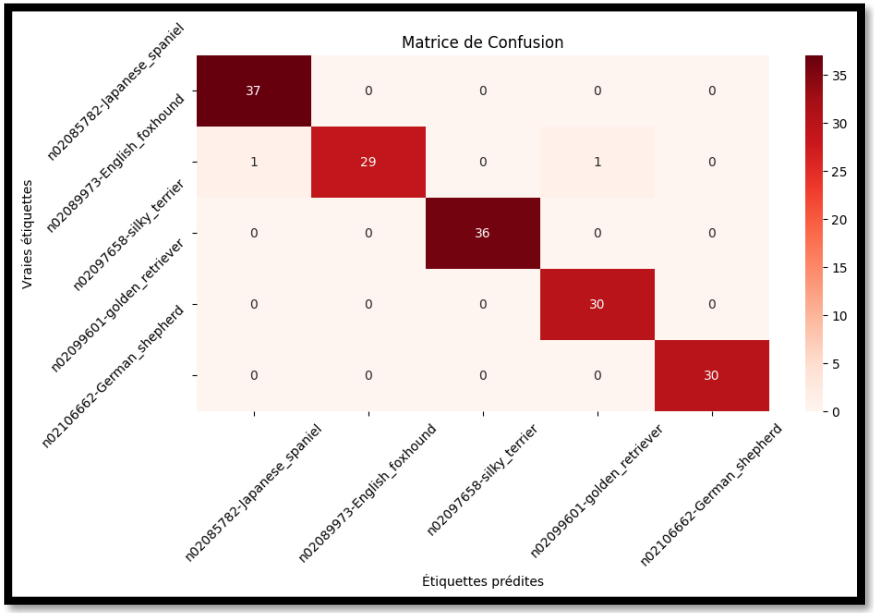


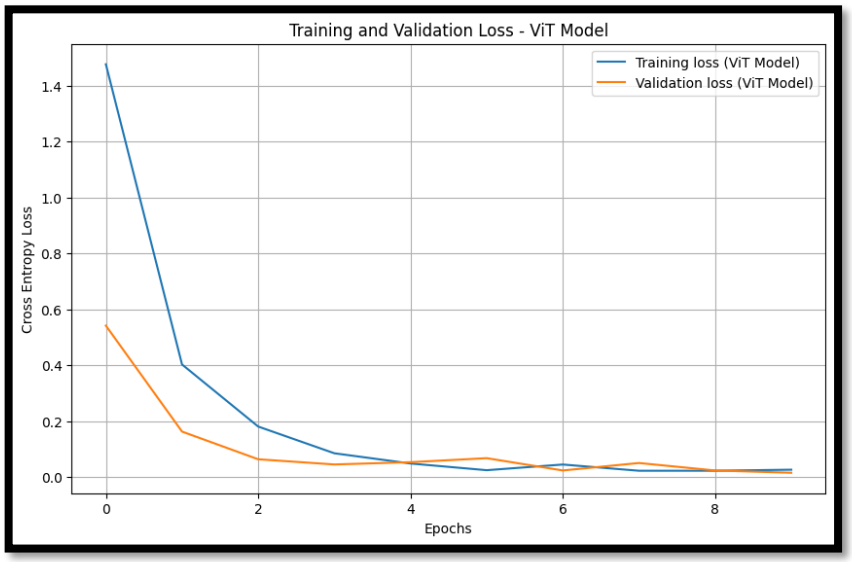
Modèle ViT (Vision Transformer)

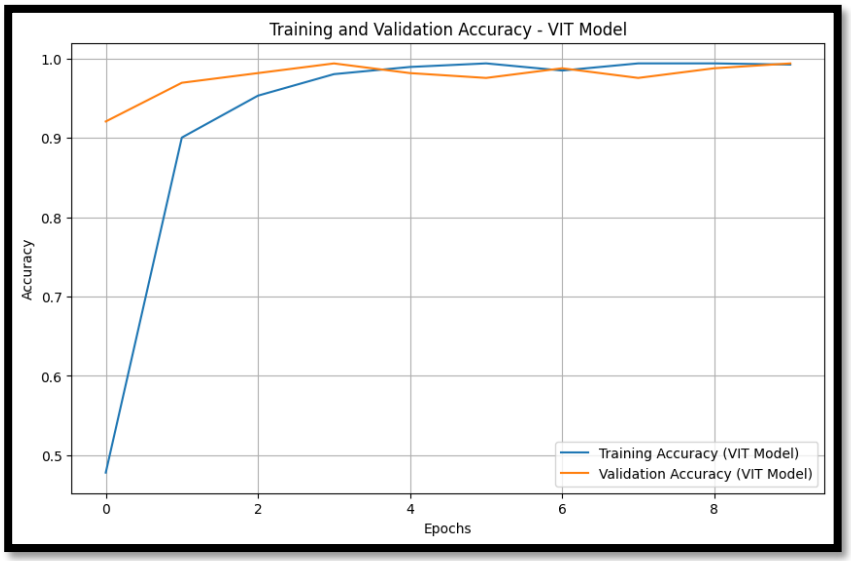
ViT en transfer learning: Résultats

Résultats:

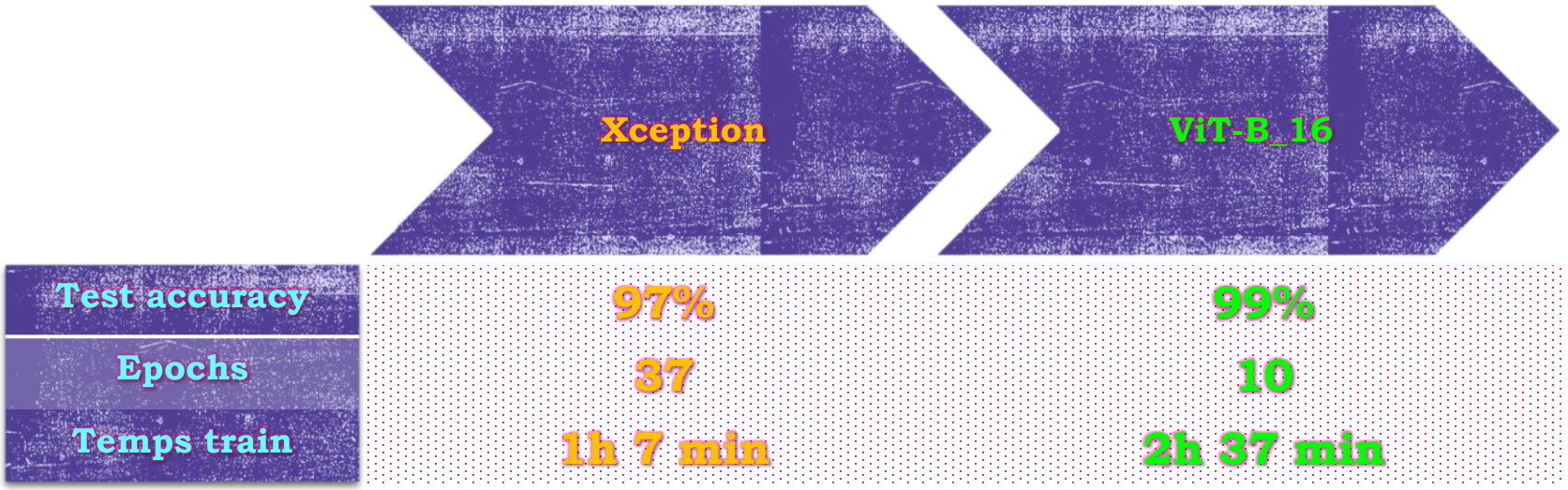
	precision	recall	f1-score	support
n02085782-Japanese_spaniel	0.97	1.00	0.99	37.00
n02089973-English_foxhound	1.00	0.94	0.97	31.00
n02097658-silky_terrier	1.00	1.00	1.00	36.00
n02099601-golden_retriever	0.97	1.00	0.98	30.00
n02106662-German_shepherd	1.00	1.00	1.00	30.00
accuracy	0.99	0.99	0.99	0.99
macro avg	0.99	0.99	0.99	164.00
weighted avg	0.99	0.99	0.99	164.00







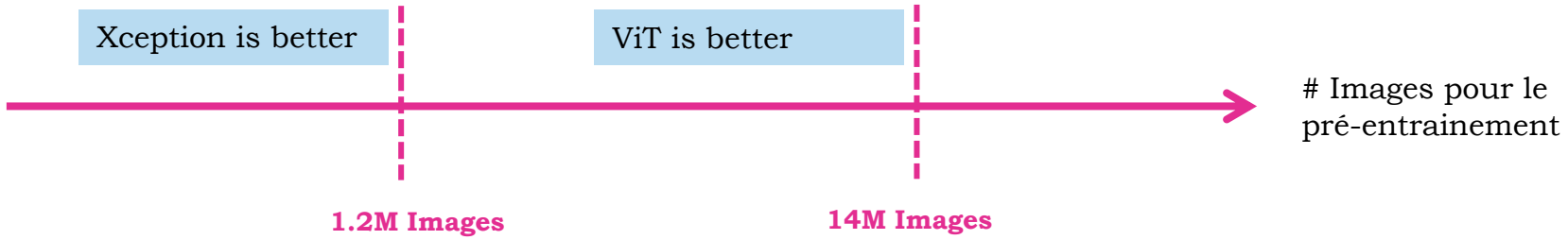
Sélection du meilleur modèle



Les résultats montrent que **ViT** dépasse **Xception** en précision avec **99%** sur l'ensemble de test, contre **97%** pour Xception. ViT nécessite moins d'epochs (10 vs. 37) mais un temps d'entraînement plus long (2h 37min contre 1h 7min), indiquant des différences en efficacité des ressources d'entraînement.

Comparaison entre modèle Xception (CNN) et ViT (Transformers)

Critères	Modèle Xception (CNN)	Modèle ViT (ViT-B_16 Transformers)
Performance	• Atteint jusqu'à **97 %** de **précision** après fine-tuning (contre 96 % dans le projet 6)	• Atteint **99 %** de **précision** sur **ViT-B_16**
Jeu de données pré-entraînement	• **ImageNet-1K** (**1 000 classes**)	• **ImageNet-21K** (**21 000 classes**) pour pré-entraînement, ImageNet-1K pour fine-tuning
Avantages	• Efficace pour des tâches de **classification classique** • Optimisé pour des **détails locaux** sur des jeux de données plus petits	• **Capture les relations globales** entre différentes parties de l'image • Optimisé pour les tâches complexes nécessitant une compréhension globale
Inconvénients	• Nécessite **plus de données** pour généraliser correctement • Ressources **computationnelles élevées** (GPU, mémoire)	• Moins performant pour **les détails locaux** • Nécessite également **plus de ressources computationnelles** pour de plus grands ensembles de données
Type de traitement	• Utilise des **convolutions** pour extraire des caractéristiques locales	• Utilise des **patches d'images** et des **mécanismes d'attention** pour capturer des relations globales

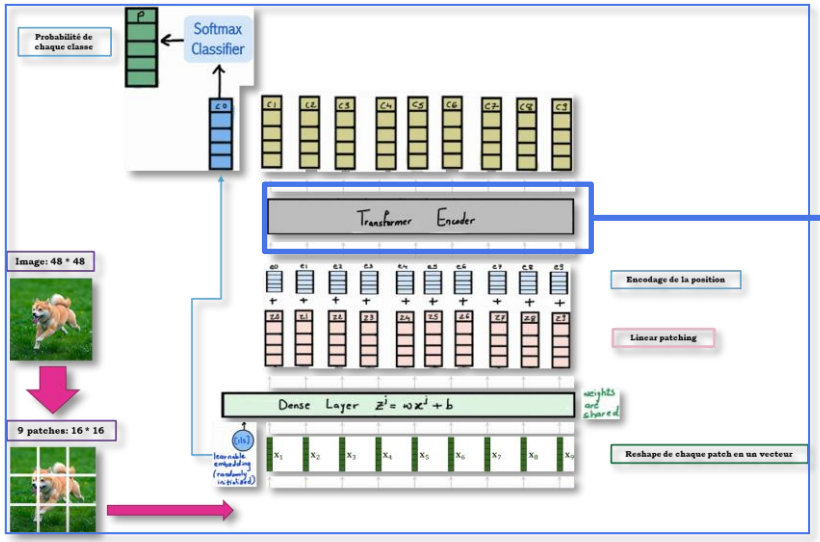


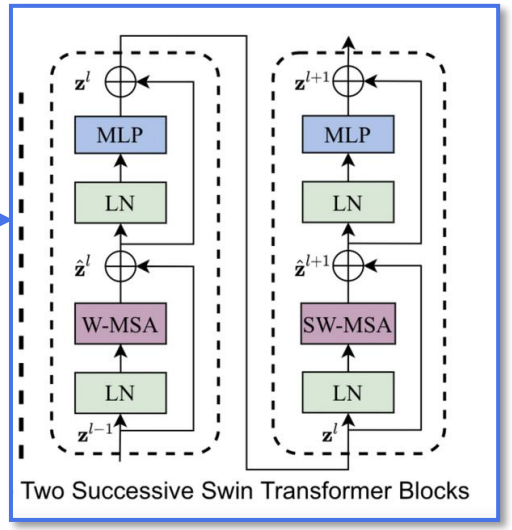
- **Conclusion:** Xception est idéal pour des tâches de classification locales et plus conventionnelles, tandis que ViT excelle dans les tâches nécessitant une analyse plus globale et contextuelle.

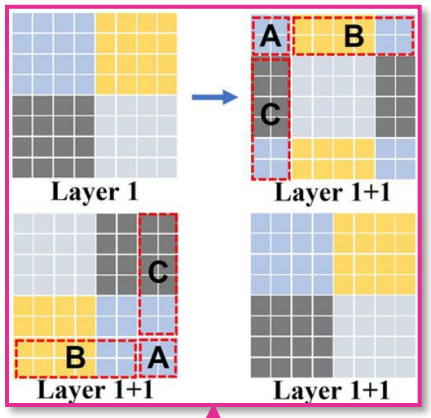
Benchmark des nouveaux modèles

Swin Transformers : Présentation

Le Swin Transformer suit des **étapes similaires** à celles du **ViT**, à **l'exception** de l'utilisation d'un **encoder** spécifique avec **des fenêtres glissantes** (sliding windows) au lieu de l'encoder **transformer classique**.







- **Layer 1 :** Applique une **attention locale** à des fenêtres non chevauchantes, ce qui est efficace en termes de calcul.
- **Layer 1+1 (Shifted Windows)** : Les fenêtres sont glissées pour couvrir des parties voisines de l'image, permettant au modèle de **capturer des relations globales** tout en **conservant une approche locale** dans la première étape.

Xception is better | ViT is better | Swin is better

Images pour le pré-entrainement

- ❑ 1.2M Images
- ❑ ImageNet-1K

- ❑ 14M Images
- ❑ ImageNet-21K

- ❑ 14M Images
- ❑ ImageNet-22K

Conclusion: Swin Transformer **combine** les **avantages** de **la capture locale et globale**, ce qui le rend particulièrement **adapté aux tâches complexes** tout en restant plus **efficace** en **termes de calcul** que le **ViT**.

Conclusion

❑ Constats :

- ✓ Le modèle **Xception** a atteint **97 %** de précision après finetuning, prouvant son efficacité pour la classification d'images.
- ✓ Le **ViT-B16** a montré une précision de **99 %**, excellent pour les tâches nécessitant une **compréhension globale** de l'image.
- ✓ Le **ViT from scratch** n'a pas bien performé, principalement à cause du petit dataset de **827 images** réparties sur **5 classes**, insuffisant pour un modèle aussi complexe.
- ✓ Suite à un benchmark, le **Swin Transformer**, identifié comme une **évolution du ViT**, est plus performant et adaptable grâce à sa combinaison d'attention locale et globale, bien que son implémentation soit plus complexe à réaliser dans le notebook.

❑ Pistes d'amélioration :

- ➢ **Augmentation du dataset** pour améliorer les performances, surtout pour le **ViT from scratch**.
- ➢ **Exploration de modèles hybrides (CNN + Transformers)**, comme les **Convolutional Vision Transformers (CvT)**, pour combiner les avantages des deux approches.
- ➢ **Tester le modèle Swin en transfert learning** dans le notebook pour exploiter pleinement ses capacités sur ce dataset.