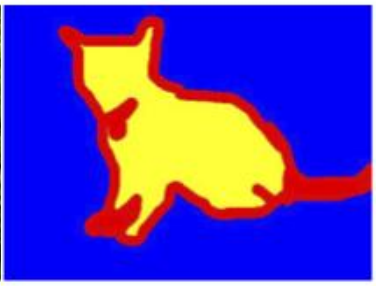
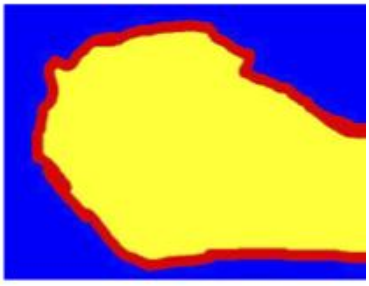
TP U-NET

1. Chargement et mise en forme des données

On utilisera une base de données annotée composée de chiens et de chats de différentes espèces provenant de oxford https://www.robots.ox.ac.uk/~vgg/data/pets/
Elle est composée de 7349 images de tailles différentes ainsi que de 'trimaps' séparant les objets du fond/



Charger les données en utilisant les commandes suivantes :

```
X2 = np.load("data64_red.npy")
Y2 = np.load("mask64_red.npy")

X_train, X_test, Y_train, Y_test = train_test_split(X2, Y2, test_size=0.2, random_state=42)
```

Questions

- Essayer de deviner quels traitements ont été réalisés sur les images et les masques d'annotations pour arriver à X2 et Y2 (on comparera les images et les masques à ceux donnés dans le répertoire du TP en termes de dynamique, de taille,…).
- Quel est le pourcentage de données mis dans X_train, X_test ? Combien y a-t-il d'images en train et en test ?

2. Définition et test d'un U-Net très simple

2.1. Définition du réseau

On va utiliser un réseau très simple :

```
def unet_simple(input_shape):
    inputs = layers.Input(input_shape)

    # Encodeur
    c1 = layers.Conv2D(XXX, (3, 3), activation='relu', padding='same', name='c1')(inputs)
    p1 = layers.MaxPooling2D((2, 2), name='p1')(c1)
```

```
    # Bottleneck
    b = layers.Conv2D(XXX, (3, 3), activation='relu', padding='same', name='b')(p1)

    # Decodeur
    u1 = layers.Conv2DTranspose(XXX, (2, 2), strides=(2, 2), padding='same', name='u1')(b)
    u1 = layers.concatenate([u1, c1], name='u1b')
    cd1 = layers.Conv2D(XXX, (3, 3), activation='relu', padding='same', name='cd1')(u1)

    outputs = layers.Conv2D(1, (1, 1), activation='XXXX', name='outputs')(cd1)
    model = models.Model(inputs, outputs)
    return model
```

Remplacer tous les XXX par les variables ou valeurs adéquates de manière à avoir une profondeur de 128 au bottleneck.

Afficher le nombre de paramètres du réseau (model.summary())

2.2. Apprentissage

Afin de trouver un bon modèle, les pré-tests seront réalisés avec **10 epochs**.

Apprendre le modèle avec XXX pour fonction de perte et **l'accuracy** pour métrique. Rappeler ce que représentent ces deux notions. On utilisera l'optimiseur Adam, une **taille de batch de 8** et un **pas d'apprentissage par défaut de 0.0001**. Mesurer le temps moyen mis par epoch. Quelle est l'accuracy sur la base de test ?

```
ad= Adam(learning_rate=lr)
model.compile(optimizer = ad, loss= XXX, metrics=['accuracy'])
tps1 = time.time()
history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size, validation_data=(X_test, Y_test))
tps2 = time.time()
affiche(history)
```

On affichera l'évolution des fonctions de perte et accuracy avec

```
def affiche(history):
    # summarize history for accuracy
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()
    # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()
```

On complétera le tableau suivant au fur et à mesure des tests réalisés.

	Fn perte	profondeur	epoch	temps	Nb param	Accuracy
	cross-entropie binaire	1	10			

2.3. Prédiction

Réaliser la prédiction des exemples de la base de test et afficher quelques résultats en utilisant la fonction display_results() ci-dessous.

```
def display_results(X_test, Y_test, preds):
    plt.figure(figsize=(12, 12))
    for i in range(5):
        plt.subplot(5, 3, 3*i+1)
        plt.imshow(X_test[i], cmap='gray')
        plt.title("Image")

        plt.subplot(5, 3, 3*i+2)
        plt.imshow(Y_test[i], cmap='gray')
        plt.title("True Mask")

        plt.subplot(5, 3, 3*i+3)
        plt.imshow(preds[i]>0.5, cmap='gray')
        plt.title("Predicted Mask")

    plt.tight_layout()
    plt.show()
```

3. U-Net plus profond et plus complexe

3.1. U-Net avec une profondeur de 3

Reprendre le réseau précédent en ajoutant 2 étages (résolutions) supplémentaires. On réglera le nombre de filtres de chaque étage de manière à conserver une profondeur de 128 au bottleneck. Que constatez-vous sur les résultats ?

3.2. U-Net avec deux couches de convolutions par étage

On va maintenant mettre 2 couches convolutionnelles à chaque étage de l'encodeur et du décodeur. Comment évoluent les résultats ?
Gardez ce modèle s'il améliore les performances, sinon, revenez au précédent.

3.1. U-Net avec du UpSampling2D à la place des Conv2DTranspose

On va maintenant remplacer les convolutions transposées par de l'upsampling dans le décodeur. Comment évoluent les résultats ?
Gardez ce modèle s'il améliore les performances, sinon, revenez au précédent.

3.2. U-Net avec plus ou moins de filtres

Faites varier la profondeur au bottleneck (64, 128, 256) et en conséquence le nombre de filtres dans l'encodeur et le décodeur. Etudiez l'évolution des résultats et des temps de calcul. Pour la suite, on gardera une profondeur de 128.

3.3. U-Net appris avec la fonction perte Dice

Rappeler ce qu'est la fonction perte Dice. Relancer l'apprentissage avec cette fonction perte. Conclusion
Gardez ce modèle s'il améliore les performances, sinon, revenez au précédent.

4. Deep supervision

4.1. U-Net appris avec de la supervision profonde

On souhaite essayer la deep supervision pour voir si elle améliore l'apprentissage. Pour cela, une carte de segmentation est produite à la sortie de chaque étage (de la même façon que pour le dernier étage) et le réseau produira 3 sorties :

```
model = models.Model(inputs, [output1, output2, output3])
```

La fonction perte et la métrique seront définies pour chaque sortie :

```
model.compile(
   optimizer="adam",
   loss={
      "output1": "binary_crossentropy",
      "output2": "binary_crossentropy",
      "output3": "binary_crossentropy"
   },
   loss_weights={
      "output1": 1,
      "output2": 1,
      "output3":1
   },
   metrics={
      "output1": ["accuracy"],
      "output2": ["accuracy"],
      "output3": ["accuracy"]
   }
)
```

Et les sorties Y_train et Y_test devront être redéfinies pour correspondre au nouveau problème. On écrira pour cela une fonction Y_new = prepare_sortie(Y).

Pousser l'apprentissage avec plus d'epochs pour obtenir les résultats finaux.

5. Test sur une image réelle

Donner le masque de segmentation de l'image Abyssinian_1.jpg, puis de l'image yorkshire_terrier_200.jpg.

Commenter.