CSE463 LAB FINAL CHEAT SHEET

PLEASE BRING A PRINTED COPY OF THIS SHEET IF YOU NEED IT DURING YOUR EXAM. YOU WILL NOT BE ALLOWED TO ACCESS THE INTERNET DURING THE EXAM.

1. Conv2D: Applies a 2D convolution to the input tensor.

- filters: *(int)* Number of output filters.
- kernel_size: *(tuple or int)* Height and width of the convolution kernel.
- activation: *(str)* Activation function (e.g., 'relu').
- padding: *(str)* Padding mode, either 'same' or 'valid'.

Usage: layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same')(x)

2. Conv2DTranspose: Applies a transposed convolution (deconvolution) operation to the input tensor.

- filters: *(int)* Number of output filters.
- kernel_size: *(tuple or int)* Height and width of the convolution kernel.
- strides: *(tuple or int)* Strides of the convolution.
- padding: *(str)* Padding mode, either 'same' or 'valid'.
- activation: *(str)* Activation function (e.g., 'relu').

Usage: layers.Conv2DTranspose(filters=64, kernel_size=(3, 3), strides=2, padding='same', activation='relu')(x)

3. MaxPooling2D: Applies a max-pooling operation to reduce the spatial dimensions of the input tensor.

- pool_size: *(tuple or int)* Size of the pooling window (e.g., (2, 2)).
- strides: *(tuple or int)* Stride of the pooling operation.
- padding: *(str)* Padding mode, either 'same' or 'valid'.

Usage: layers.MaxPooling2D((2, 2), strides=(2, 2), padding="same")(x)

4. concatenate: combines multiple tensors along a specified axis (default is the last axis).

- **inputs**: A list of tensors (e.g., [tensor1, tensor2, ...]) to concatenate.
- **axis**: An integer specifying the axis along which to concatenate. The default is -1 (last axis).

Usage:

```
u9 = layers.Conv2D(64, (3, 3), padding="same", activation="relu")(some_input)
c1 = layers.Conv2D(64, (3, 3), padding="same", activation="relu")(another_input)
merged = layers.concatenate([u9, c1])
```

5. Input: Defines the shape of the input tensor for the model.

- shape: *(tuple)* Shape of the input tensor (e.g., (128, 128, 3)).

Usage: layers.Input(shape=(128, 128, 3))

6. Model: Combines inputs and outputs into a complete model.

- inputs: *(Tensor)* Input tensor(s) of the model.
- outputs: *(Tensor)* Output tensor(s) of the model.
- name: *(str)* Optional name of the model.

Usage: Model(inputs=input_tensor, outputs=output_tensor, name="U-Net")

7. BatchNormalization: Normalizes the outputs of the previous layer to stabilize and accelerate training.

Usage: layers.BatchNormalization()

8. ReLU or LeakyReLU: Applies the Rectified Linear Unit (ReLU) activation function.

Usage: layers.ReLU() or layers.LeakyReLU()

9. Dense: Fully connected layer that computes the dot product of inputs and weights, followed by an optional activation function.

- units: *(int)* Number of neurons in the layer.
- activation: *(str or callable)* Activation function to use (e.g., 'relu', 'sigmoid').
- use_bias: *(bool)* Whether to include a bias term in the layer.
- kernel_initializer: *(str or initializer)* Initializer for the weights.
- bias_initializer: *(str or initializer)* Initializer for the bias.

Usage: layers.Dense(units=128, activation='relu', use_bias=True) OR
layers.Dense(7*7*256, use_bias=False, input_shape=(100,))

10. UpSampling2D: Upsamples the input tensor along its spatial dimensions by repeating values.

- size: *(tuple or int)* Upsampling factor for height and width (e.g., (2, 2)).
- interpolation: *(str)* Method for interpolation ('nearest', 'bilinear').

Usage: layers.UpSampling2D(size=(2, 2), interpolation='nearest')

11. Flatten: Flattens the input tensor to a 1D vector, often used before feeding into a dense layer.

Usage: layers.Flatten()

12. Dropout: Applies dropout regularization by randomly setting a fraction of input units to 0 at each update during training.

- rate: *(float)* Fraction of the input units to drop (e.g., 0.5 for 50%).

Usage: layers.Dropout(rate=0.5)

13. GlobalAveragePooling2D: Computes the global average of each feature map, reducing the spatial dimensions to 1.

Usage: layers.GlobalAveragePooling2D()

14. Add: Adds layers to the model stack.

Usage: model.add(layers.GlobalAveragePooling2D())

15. Compile: Compiles the final model for execution.

Usage: model.compile(optimizer='adam', loss='mean_squared_error',
metrics=['MeanSquaredError', 'AUC',])

16. BinaryCrossentropy:.Computes the cross-entropy loss between true labels and predicted labels.

Usage:
```
cross_entropy = cross_entropy = tf.keras.losses.BinaryCrossentropy()
loss = cross_entropy(tf.ones_like(fake_output), fake_output)
```