Describing UI Screenshots in Natural Language
Supplementary Materials

Luis A. Leiva, Asutosh Hota, Antti Oulasvirta

This document provides supplementary information about the different design, development, and implementation choices of the different components of XUI's processing pipeline. In our research paper, several UI input formats and models have been tested, together with their combinations. The following document provides more details about those models and their architectures.

Figure 1 shows a few examples of the evaluated UI screenshots, their corresponding semantic wireframes, and the reconstructions of the semantic wireframes using a deep convolutional autoencoder trained on the Enrico dataset [3], which is a curated version of the Rico dataset [4]. The encoder part, shown in Figure 2, takes as input a 256x128x3 dimensional UI wireframe (RGB-based image) and generates a 32x16x32 dimensional UI embedding as output. This embedding is later used along with other UI inputs (RGB screenshots and semantic wirerames) to try out how different model configurations perform in the Topic classification task.

Figures 3 to 6 show the different combinations of UI inputs and their respective model architectures that we tested, aimed at finding the best possible combination for the topic classification task. The individual models (not show in this report, for brevity's sake) can be noticed in each of the individual branches of the combined models.

To find the visual saliency of the UI screenshot, we use a gradient-based localization technique [5] to compute the salient regions that our ConvNet looks at during topic classification. The backbone arhcitecture used for this purpose is based on the popular VGG16 architecture [6]. We compared this saliency model against other comparable approaches, namely GBVS [1] and the feature maps produced by ResNet50 [2]. GBVS is a classical model that requires no training for prediction of saliency regions in an image and achieves reasonable performance. It works in two stages, first forming activation maps on certain feature channels, and then normalizing them in a way which highlights conspicuity and admits combination with other maps [1]. On the other hand, ResNet is a deep learning model used in image classification tasks, very popular because of its generalization capabilities (thanks to transfer learning). Figure 7 shows its architecture. There are 5 blocks, each with an identity and a convolutional block. Each block consists of 3 convolutional layers.

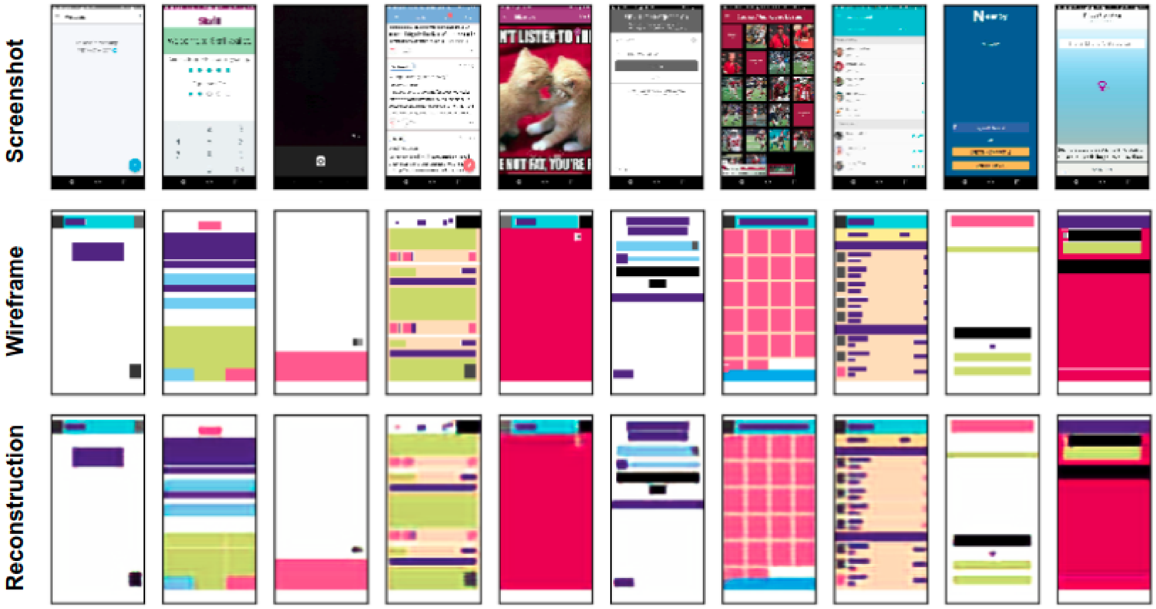

Figure 1: Reconstructions of semantic wireframes using the model depicted in Figure 2. The dataset provides semantic wireframes for their corresponding original screenshot, which inform about the structure of the UI.

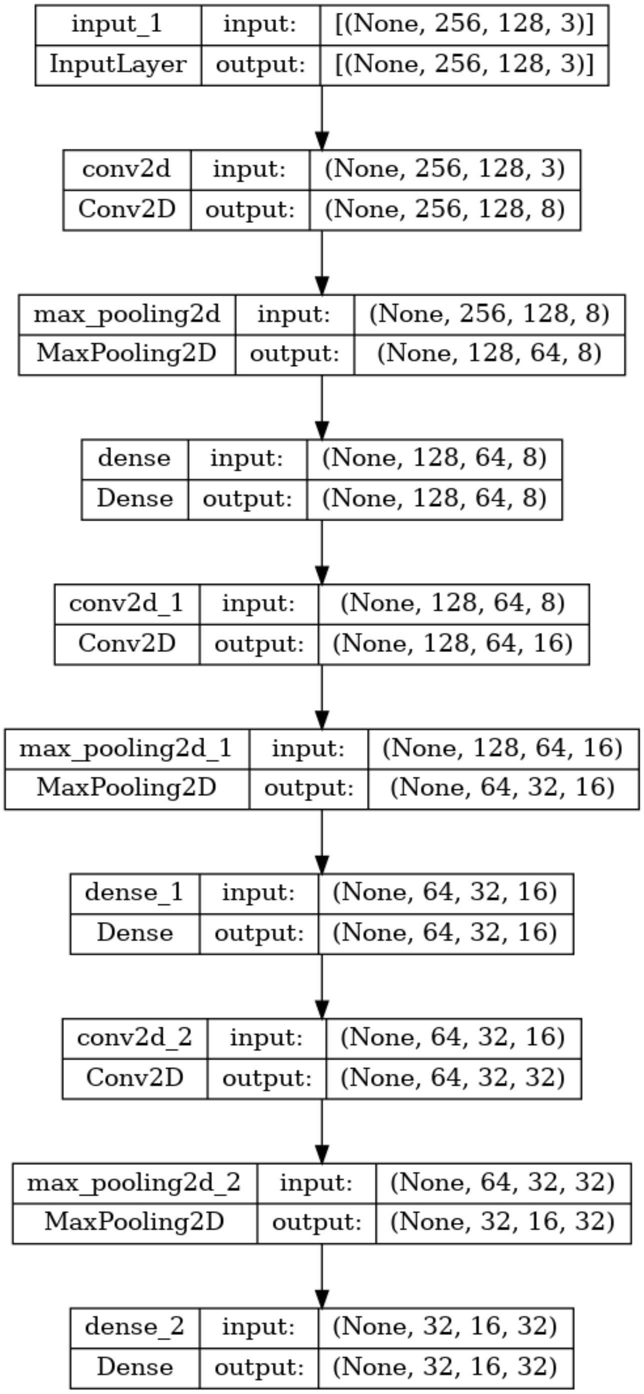
input_1 | input: | [(None, 256, 128, 3)]
InputLayer | output: | [(None, 256, 128, 3)]
conv2d | input: | (None, 256, 128, 3)
Conv2D | output: | (None, 256, 128, 8)
max_pooling2d | input: | (None, 256, 128, 8)
MaxPooling2D | output: | (None, 128, 64, 8)
dense | input: | (None, 128, 64, 8)
Dense | output: | (None, 128, 64, 8)
conv2d_1 | input: | (None, 128, 64, 8)
Conv2D | output: | (None, 128, 64, 16)
max_pooling2d_1 | input: | (None, 128, 64, 16)
MaxPooling2D | output: | (None, 64, 32, 16)
dense_1 | input: | (None, 64, 32, 16)
Dense | output: | (None, 64, 32, 16)
conv2d_2 | input: | (None, 64, 32, 16)
Conv2D | output: | (None, 64, 32, 32)
max_pooling2d_2 | input: | (None, 64, 32, 32)
MaxPooling2D | output: | (None, 32, 16, 32)
dense_2 | input: | (None, 32, 16, 32)
Dense | output: | (None, 32, 16, 32)

Figure 2: Encoder architecture to generate semantic embeddings from UI wireframes.

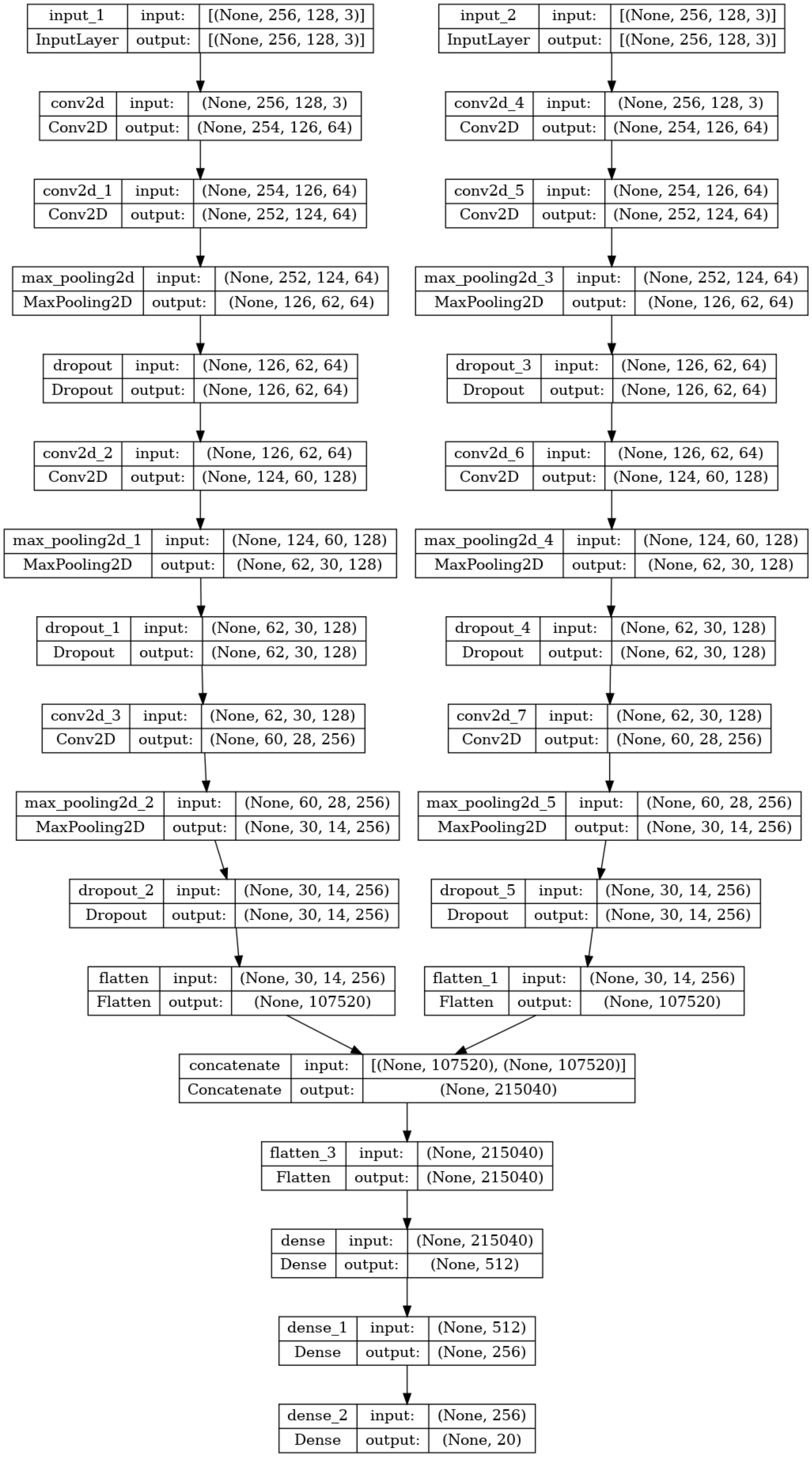
input_1 | InputLayer | input: [(None, 256, 128, 3)] | output: [(None, 256, 128, 3)]
conv2d | Conv2D | input: (None, 256, 128, 3) | output: (None, 254, 126, 64)
conv2d_1 | Conv2D | input: (None, 254, 126, 64) | output: (None, 252, 124, 64)
max_pooling2d | MaxPooling2D | input: (None, 252, 124, 64) | output: (None, 126, 62, 64)
dropout | Dropout | input: (None, 126, 62, 64) | output: (None, 126, 62, 64)
conv2d_2 | Conv2D | input: (None, 126, 62, 64) | output: (None, 124, 60, 128)
max_pooling2d_1 | MaxPooling2D | input: (None, 124, 60, 128) | output: (None, 62, 30, 128)
dropout_1 | Dropout | input: (None, 62, 30, 128) | output: (None, 62, 30, 128)
conv2d_3 | Conv2D | input: (None, 62, 30, 128) | output: (None, 60, 28, 256)
max_pooling2d_2 | MaxPooling2D | input: (None, 60, 28, 256) | output: (None, 30, 14, 256)
dropout_2 | Dropout | input: (None, 30, 14, 256) | output: (None, 30, 14, 256)
flatten | Flatten | input: (None, 30, 14, 256) | output: (None, 107520)
input_2 | InputLayer | input: [(None, 256, 128, 3)] | output: [(None, 256, 128, 3)]
conv2d_4 | Conv2D | input: (None, 256, 128, 3) | output: (None, 254, 126, 64)
conv2d_5 | Conv2D | input: (None, 254, 126, 64) | output: (None, 252, 124, 64)
max_pooling2d_3 | MaxPooling2D | input: (None, 252, 124, 64) | output: (None, 126, 62, 64)
dropout_3 | Dropout | input: (None, 126, 62, 64) | output: (None, 126, 62, 64)
conv2d_6 | Conv2D | input: (None, 126, 62, 64) | output: (None, 124, 60, 128)
max_pooling2d_4 | MaxPooling2D | input: (None, 124, 60, 128) | output: (None, 62, 30, 128)
dropout_4 | Dropout | input: (None, 62, 30, 128) | output: (None, 62, 30, 128)
conv2d_7 | Conv2D | input: (None, 62, 30, 128) | output: (None, 60, 28, 256)
max_pooling2d_5 | MaxPooling2D | input: (None, 60, 28, 256) | output: (None, 30, 14, 256)
dropout_5 | Dropout | input: (None, 30, 14, 256) | output: (None, 30, 14, 256)
flatten_1 | Flatten | input: (None, 30, 14, 256) | output: (None, 107520)
concatenate | Concatenate | input: [(None, 107520), (None, 107520)] | output: (None, 215040)
flatten_3 | Flatten | input: (None, 215040) | output: (None, 215040)
dense | Dense | input: (None, 215040) | output: (None, 512)
dense_1 | Dense | input: (None, 512) | output: (None, 256)
dense_2 | Dense | input: (None, 256) | output: (None, 20)

Figure 3: Model architecture for the combined input of Screenshot and Wireframe (denoted as $\mathcal{S}+\mathcal{W}$ in our paper).

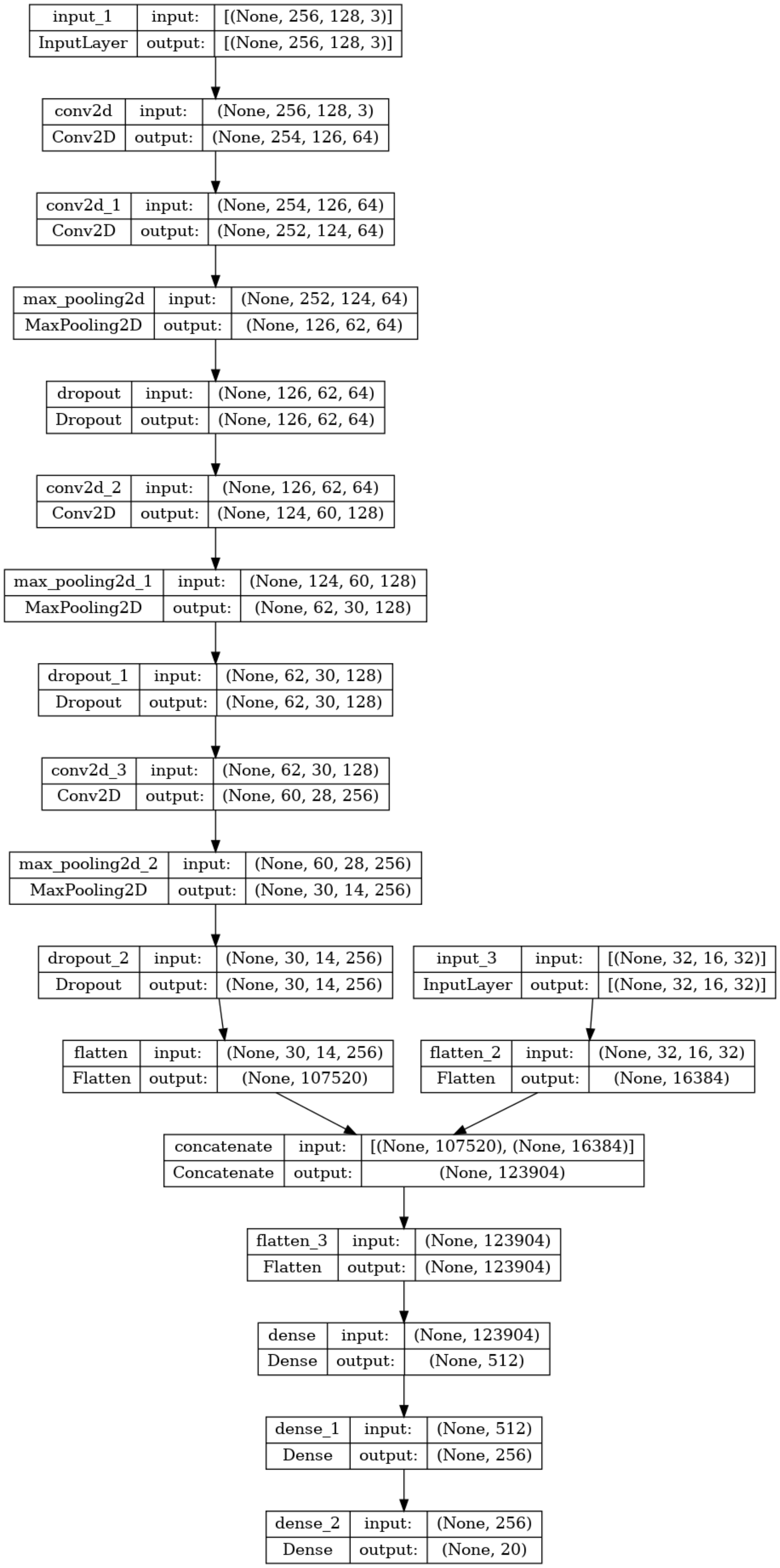
input_1 | input: | [(None, 256, 128, 3)]
InputLayer | output: | [(None, 256, 128, 3)]
conv2d | input: | (None, 256, 128, 3)
Conv2D | output: | (None, 254, 126, 64)
conv2d_1 | input: | (None, 254, 126, 64)
Conv2D | output: | (None, 252, 124, 64)
max_pooling2d | input: | (None, 252, 124, 64)
MaxPooling2D | output: | (None, 126, 62, 64)
dropout | input: | (None, 126, 62, 64)
Dropout | output: | (None, 126, 62, 64)
conv2d_2 | input: | (None, 126, 62, 64)
Conv2D | output: | (None, 124, 60, 128)
max_pooling2d_1 | input: | (None, 124, 60, 128)
MaxPooling2D | output: | (None, 62, 30, 128)
dropout_1 | input: | (None, 62, 30, 128)
Dropout | output: | (None, 62, 30, 128)
conv2d_3 | input: | (None, 62, 30, 128)
Conv2D | output: | (None, 60, 28, 256)
max_pooling2d_2 | input: | (None, 60, 28, 256)
MaxPooling2D | output: | (None, 30, 14, 256)
dropout_2 | input: | (None, 30, 14, 256)
Dropout | output: | (None, 30, 14, 256)
input_3 | input: | [(None, 32, 16, 32)]
InputLayer | output: | [(None, 32, 16, 32)]
flatten | input: | (None, 30, 14, 256)
Flatten | output: | (None, 107520)
flatten_2 | input: | (None, 32, 16, 32)
Flatten | output: | (None, 16384)
concatenate | input: | [(None, 107520), (None, 16384)]
Concatenate | output: | (None, 123904)
flatten_3 | input: | (None, 123904)
Flatten | output: | (None, 123904)
dense | input: | (None, 123904)
Dense | output: | (None, 512)
dense_1 | input: | (None, 512)
Dense | output: | (None, 256)
dense_2 | input: | (None, 256)
Dense | output: | (None, 20)

Figure 4: Model architecture for the combined input of Screenshot and Embedding (denoted as $\mathcal{S} + \mathcal{E}$ in our paper).

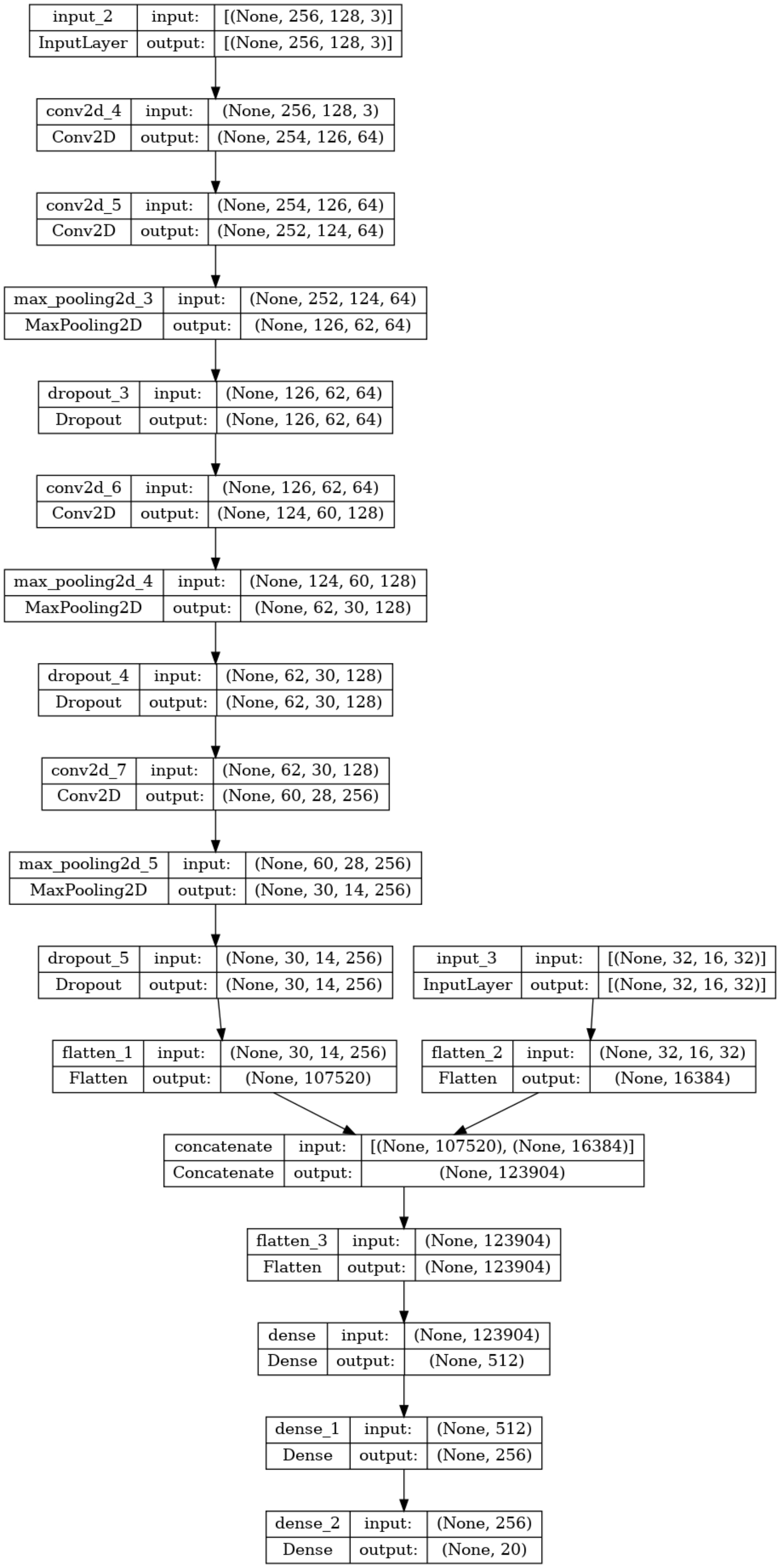
input_2 | input: | [(None, 256, 128, 3)]
InputLayer | output: | [(None, 256, 128, 3)]
conv2d_4 | input: | (None, 256, 128, 3)
Conv2D | output: | (None, 254, 126, 64)
conv2d_5 | input: | (None, 254, 126, 64)
Conv2D | output: | (None, 252, 124, 64)
max_pooling2d_3 | input: | (None, 252, 124, 64)
MaxPooling2D | output: | (None, 126, 62, 64)
dropout_3 | input: | (None, 126, 62, 64)
Dropout | output: | (None, 126, 62, 64)
conv2d_6 | input: | (None, 126, 62, 64)
Conv2D | output: | (None, 124, 60, 128)
max_pooling2d_4 | input: | (None, 124, 60, 128)
MaxPooling2D | output: | (None, 62, 30, 128)
dropout_4 | input: | (None, 62, 30, 128)
Dropout | output: | (None, 62, 30, 128)
conv2d_7 | input: | (None, 62, 30, 128)
Conv2D | output: | (None, 60, 28, 256)
max_pooling2d_5 | input: | (None, 60, 28, 256)
MaxPooling2D | output: | (None, 30, 14, 256)
dropout_5 | input: | (None, 30, 14, 256)
Dropout | output: | (None, 30, 14, 256)
input_3 | input: | [(None, 32, 16, 32)]
InputLayer | output: | [(None, 32, 16, 32)]
flatten_1 | input: | (None, 30, 14, 256)
Flatten | output: | (None, 107520)
flatten_2 | input: | (None, 32, 16, 32)
Flatten | output: | (None, 16384)
concatenate | input: | [(None, 107520), (None, 16384)]
Concatenate | output: | (None, 123904)
flatten_3 | input: | (None, 123904)
Flatten | output: | (None, 123904)
dense | input: | (None, 123904)
Dense | output: | (None, 512)
dense_1 | input: | (None, 512)
Dense | output: | (None, 256)
dense_2 | input: | (None, 256)
Dense | output: | (None, 20)

Figure 5: Model architecture for the combined input of Wireframe and Embedding (denoted as $\mathcal{W}+\mathcal{E}$ in our paper).

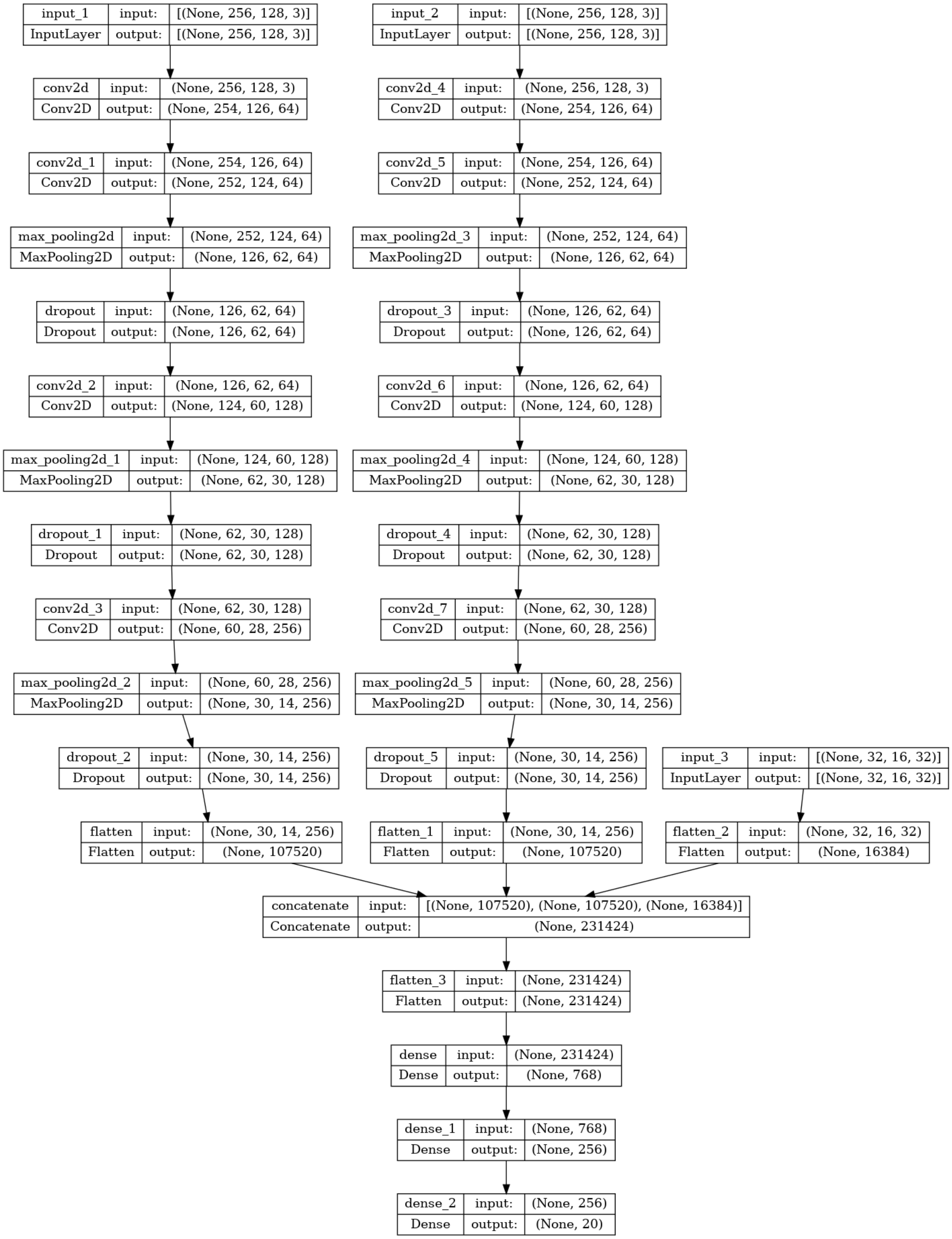
input_1 input: [(None, 256, 128, 3)]
InputLayer output: [(None, 256, 128, 3)]
input_2 input: [(None, 256, 128, 3)]
InputLayer output: [(None, 256, 128, 3)]
conv2d input: (None, 256, 128, 3)
Conv2D output: (None, 254, 126, 64)
conv2d_4 input: (None, 256, 128, 3)
Conv2D output: (None, 254, 126, 64)
conv2d_1 input: (None, 254, 126, 64)
Conv2D output: (None, 252, 124, 64)
conv2d_5 input: (None, 254, 126, 64)
Conv2D output: (None, 252, 124, 64)
max_pooling2d input: (None, 252, 124, 64)
MaxPooling2D output: (None, 126, 62, 64)
max_pooling2d_3 input: (None, 252, 124, 64)
MaxPooling2D output: (None, 126, 62, 64)
dropout input: (None, 126, 62, 64)
Dropout output: (None, 126, 62, 64)
dropout_3 input: (None, 126, 62, 64)
Dropout output: (None, 126, 62, 64)
conv2d_2 input: (None, 126, 62, 64)
Conv2D output: (None, 124, 60, 128)
conv2d_6 input: (None, 126, 62, 64)
Conv2D output: (None, 124, 60, 128)
max_pooling2d_1 input: (None, 124, 60, 128)
MaxPooling2D output: (None, 62, 30, 128)
max_pooling2d_4 input: (None, 124, 60, 128)
MaxPooling2D output: (None, 62, 30, 128)
dropout_1 input: (None, 62, 30, 128)
Dropout output: (None, 62, 30, 128)
dropout_4 input: (None, 62, 30, 128)
Dropout output: (None, 62, 30, 128)
conv2d_3 input: (None, 62, 30, 128)
Conv2D output: (None, 60, 28, 256)
conv2d_7 input: (None, 62, 30, 128)
Conv2D output: (None, 60, 28, 256)
max_pooling2d_2 input: (None, 60, 28, 256)
MaxPooling2D output: (None, 30, 14, 256)
max_pooling2d_5 input: (None, 60, 28, 256)
MaxPooling2D output: (None, 30, 14, 256)
dropout_2 input: (None, 30, 14, 256)
Dropout output: (None, 30, 14, 256)
dropout_5 input: (None, 30, 14, 256)
Dropout output: (None, 30, 14, 256)
input_3 input: [(None, 32, 16, 32)]
InputLayer output: [(None, 32, 16, 32)]
flatten input: (None, 30, 14, 256)
Flatten output: (None, 107520)
flatten_1 input: (None, 30, 14, 256)
Flatten output: (None, 107520)
flatten_2 input: (None, 32, 16, 32)
Flatten output: (None, 16384)
concatenate input: [(None, 107520), (None, 107520), (None, 16384)]
Concatenate output: (None, 231424)
flatten_3 input: (None, 231424)
Flatten output: (None, 231424)
dense input: (None, 231424)
Dense output: (None, 768)
dense_1 input: (None, 768)
Dense output: (None, 256)
dense_2 input: (None, 256)
Dense output: (None, 20)

Figure 6: Model architecture for the combined input of Screenshot, Wireframe, and Embedding (denoted as $\mathcal{S} + \mathcal{W} + \mathcal{E}$ in our paper).

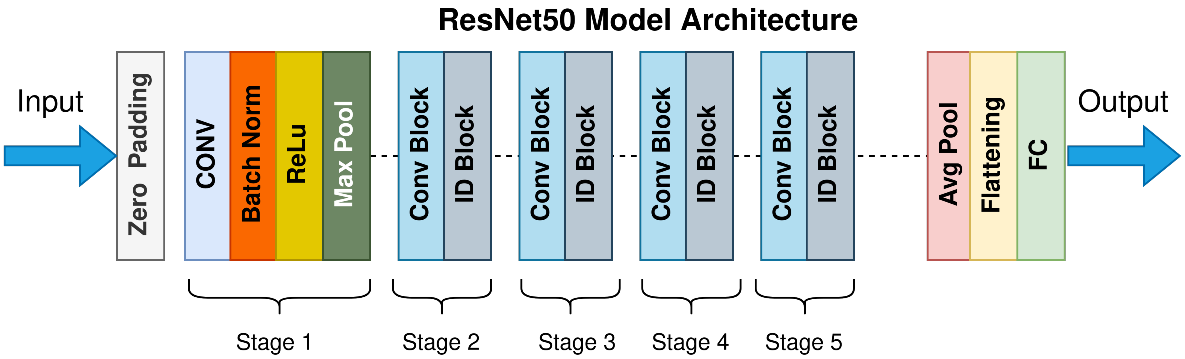

Figure 7: Model architecture of ResNet50. Source: `https://commons.wikimedia.org/wiki/File:ResNet50.png`.

References

[1] Jonathan Harel, Christof Koch, and Pietro Perona. Graph-based visual saliency. In Bernhard Schölkopf, John C. Platt, and Thomas Hofmann, editors, *Proc. NIPS*, pages 545–552, 2006.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[3] Luis A. Leiva, Asutosh Hota, and Antti Oulasvirta. Enrico: A high-quality dataset for topic modeling of mobile UI designs. In *Proc. MobileHCI*, 2020.

[4] Thomas F. Liu, Mark Craft, Jason Situ, Ersin Yumer, Radomir Mech, and Ranjitha Kumar. Learning design semantics for mobile apps. In *Proc. UIST*, pages 569–579, 2018.

[5] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-CAM: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. ICLR*, 2015.