Project Report
Denoising Document Images Using CNN-based Models

1 Introduction

Document denoising is the process of enhancing the quality and readability of document images that are degraded by noise, such as stains, blurs, shadows, or low contrast. It aims to remove the noise while preserving the text and other important information (e.g. graphics) in the document image. It is an important step for document analysis and recognition, as it can improve the accuracy and efficiency of optical character recognition (OCR) and other downstream tasks.

This project approaches a document denoising challenge on Kaggle [1] by exploring the use of different deep learning model architectures including Convolution Neural Networks, long short-term memory and attention mechanism. An image cropping method was also applied to the training data which led to significant improvements and achieved rank 8 compared to the contest leaderboard.

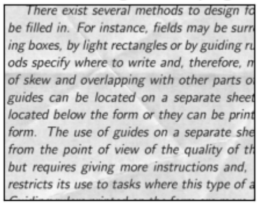
2 Background

2.1 Denoising Dirty Documents Competition

Dataset Images of either dimension 540x420 or 540x258 are provided as data. They included a training set of 144 clean images and their corresponding noised images, and 72 noised test images, in which the clean images constructed by the models are submitted and used for contest evaluation.

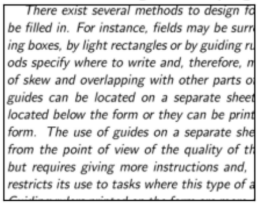
Data Sample The noises emulate real-life documents, which include coffee stains, faded sun spots, dog-eared pages, and wrinkles. Below is a sample pair of cleaned and noised training images:

Noise image: 101.png



There exist several methods to design fo
be filled in. For instance, fields may be surr
ing boxes, by light rectangles or by guiding ru
ods specify where to write and, therefore, m
of skew and overlapping with other parts o
guides can be located on a separate sheet
located below the form or they can be print
form. The use of guides on a separate she
from the point of view of the quality of th
but requires giving more instructions and,
restricts its use to tasks where this type of a

Denoised image: 101.png



There exist several methods to design fo
be filled in. For instance, fields may be surr
ing boxes, by light rectangles or by guiding ru
ods specify where to write and, therefore, m
of skew and overlapping with other parts o
guides can be located on a separate sheet
located below the form or they can be print
form. The use of guides on a separate she
from the point of view of the quality of th
but requires giving more instructions and,
restricts its use to tasks where this type of a

Scoring The score is the root mean squared error between the cleaned pixel intensities and the actual grayscale pixel intensities. The lower the score, the better the model.

2.2 Background Knowledge

A convolutional neural network (CNN) is a type of neural network that can extract local features from images or other data by applying convolutional filters.

Long short-term memory (LSTM) is a type of recurrent neural network that can capture long-term dependencies in sequential data by using memory cells and gates.

Attention is a mechanism that allows a neural network to focus on the most relevant parts of the input or output data by assigning different weights to them.

These are some deep learning architectures or structures that will be applied and evaluated in this project.

2.3 Existing Approaches

2.3.1 CNN Autoencoder

A CNN autoencoder is a type of neural network that can learn to compress and reconstruct images. It consists of two parts: an encoder and a decoder. The encoder uses convolutional layers to reduce the dimensionality of the input image and extract its features. The decoder uses transposed convolutional layers to increase the dimensionality of the encoded image and reconstruct the original image. An implementation using Keras can be found in its official documentation [2].

2.3.2 CNN-LSTM

A CNN-LSTM encoder decoder with direct attention was proposed by Haque et al. (2018) in their paper "Image denoising and restoration with CNN-LSTM Encoder Decoder with Direct Attention" [3]. The model consists of two parts: an encoder that uses a convolutional neural network (CNN) to extract features from the input image, and a decoder that uses a long short-term memory network (LSTM) to generate a clean image from the encoded features and the corrupted input. The model also uses direct attention, which allows the decoder to focus on specific regions of the input image that are relevant for the reconstruction. In the paper, the model was trained and tested on a set of corrupted MNIST digits.

3 Method

3.1 Environment

The implementation and training were performed on Kaggle, trained on its GPU P100. Keras was used to build, train and test the models. Whereas the major library used for data preprocessing is numpy and opencv.

3.2 Data Preparation

3.2.1 Image Preprocessing

The standard preprocessing procedures include conversion to grayscale, normalization (divide by 255) and reshaping to 3D array for the use of 2D convolution layers.

3.2.2 Image Cropping

A major challenge faced during training was the small set of data provided (144 images). Also, since the provided images are sections of document images, cropping them into smaller grids should not deteriorate the model's ability. Therefore, I tried to crop all the images into smaller sizes.

The cropped dimension was selected to ensure a sufficiently large grid size, so a reasonable number of lines of words are kept in each image, I also tried to minimize the padding size by selecting a new dimension to be as close to divisible by the original dimensions as possible. Since all training images are either (540x420) or (540x258), the cropped dimension (135x140) was selected.

To obtain the cropped image, the image was first centred and padded in white to get a padded image with dimensions divisible by the cropped dimension. The padded image was then cropped into grids and passed along for subsequent preprocessing and training while being treated as independent images. Note that the cleaned images are cropped in the same way so the cropped images match too.

The test images were also cropped before inferencing, and the aforementioned process was reversed the ensemble the predicted image to obtain the cleaned test image for submission.

As shown in the experiment session, this method brought a significant improvement to the model performance, so it is adapted in all other experiments.

3.2.3 Train Test Split

Due to the limited data size, a train test split was performed with 10% validation data + 90% training data.

3.3 Model Architectures

This section compares the different model architectures' performances under different numbers of training epochs.

The detail architectures and configuration of the models are included in the appendix.

I attempted to include variational autoencoder (VAE), but the model failed to converge under different configurations. After testing it on MNIST, which it converges but do not perform well, it is suspected that the lack of data and comparatively large image size makes it unable to learn under given resource constraint.

3.3.1 CNN

For CNN with cropping, the encoder contains Conv2D layers in order of 64, 256 and 128, while the decoder contains the corresponding Conv2D transpose layers to reconstruct the image.

3.3.2 CNN + LSTM

The encoder layer is the same as the CNN's encoder, except that it is flattened at the end for the LSTM input. It is then passed through 2 bidirectional LSTM layers, which outputs sequences that are reshaped back into the image.

3.3.3 CNN + Attention

The encoder is several layers of Conv2D and 2D max pooling that compresses the image into (21, 20) so the model can be trained under the resource constraint. It is passed through a multihead attention layer with 4 heads (tested to be the best value, more in experiment). Afterwards, it is passed through several Conv2d transpose and upsampling layers to reconstruct the image.

3.4 Hyperparameters

Hyperparameter	**Value**
Optimizer	adam
Loss	mean absolute error
Epochs	10/100/300
Early Callback Patience	15 epochs
Early Callback Loss	val_loss

Table 1: Hyperparameters

4 Experiments

4.1 Image Cropping

Two CNN models were trained and evaluated with or without the use of image cropping. Due to GPU resource limitation, note that the number of CNN layers in the two models is different (the model without cropping has one less layer in both the encoder and decoder part).

Epoch	**10**	**100**	**300**
without cropping	0.25488	0.04981	0.03931
with cropping	0.02683	0.0163	0.01851

Table 2: Effect of image cropping

The performance of the CNN model without image cropping performs as expected, which is around 0.04 as suggested in a tutorial notebook of the contest using CNN too (with a slightly different model architecture).

4.2 Attention Model Configuration

Number of Heads	**Result**
4	0.23373
16	0.23697
64	0.42432

Table 3: Comparison of number of heads of attention

Increasing the number of heads in the multi-head attention layer does not benefit the model's performance.

4.3 Model Architecture Evaluation

Model/Epoch	**10**	**100**	**300**
CNN	0.02683	0.0163	0.01851
CNN+LSTM	0.19022	0.07614	0.06344
CNN+Attention	0.23373	0.11954	0.09673

Table 4: Comparison of different model architectures

Unexpectedly, the simplest CNN performs significantly better. It is suspected to be caused by two reasons. Firstly, the small number of data available could limit the ability of the more complicated models to learn well. Secondly, under the GPU resource constraints, the number of layers and the encoded data sizes have to be constrained, which may also cause the models to be too shallow to learn well.

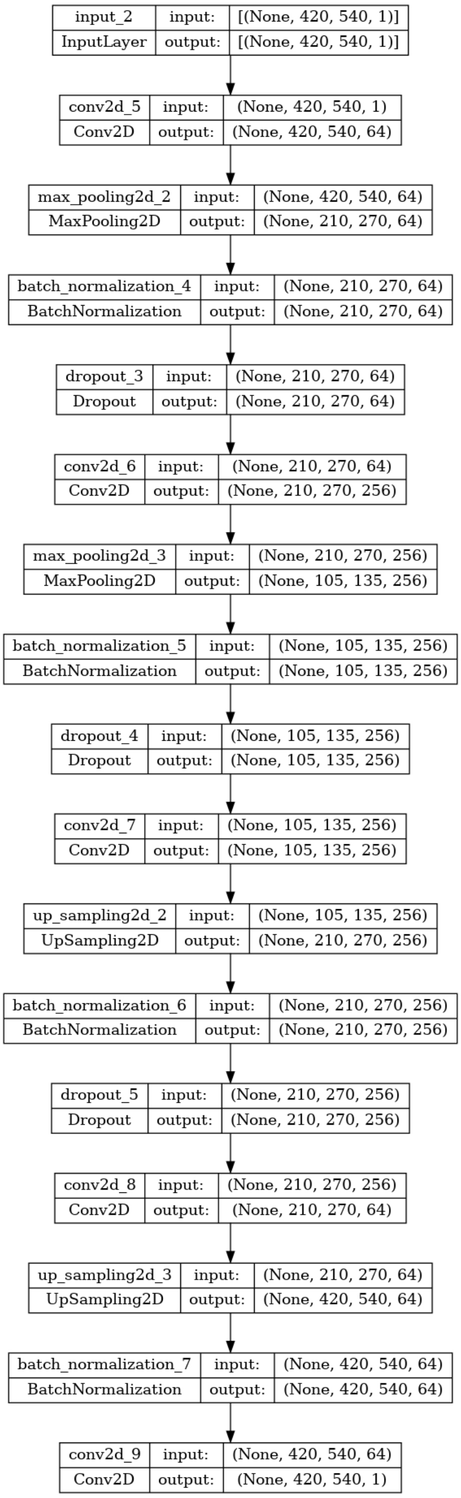
References

[1] W. Cukierski, "Denoising dirty documents." `https://kaggle.com/competitions/denoising-dirty-documents`, 2015.

[2] S. L. Valdarrama, "Convolutional autoencoder for image denoising." `https://keras.io/examples/vision/autoencoder/`, 2021.

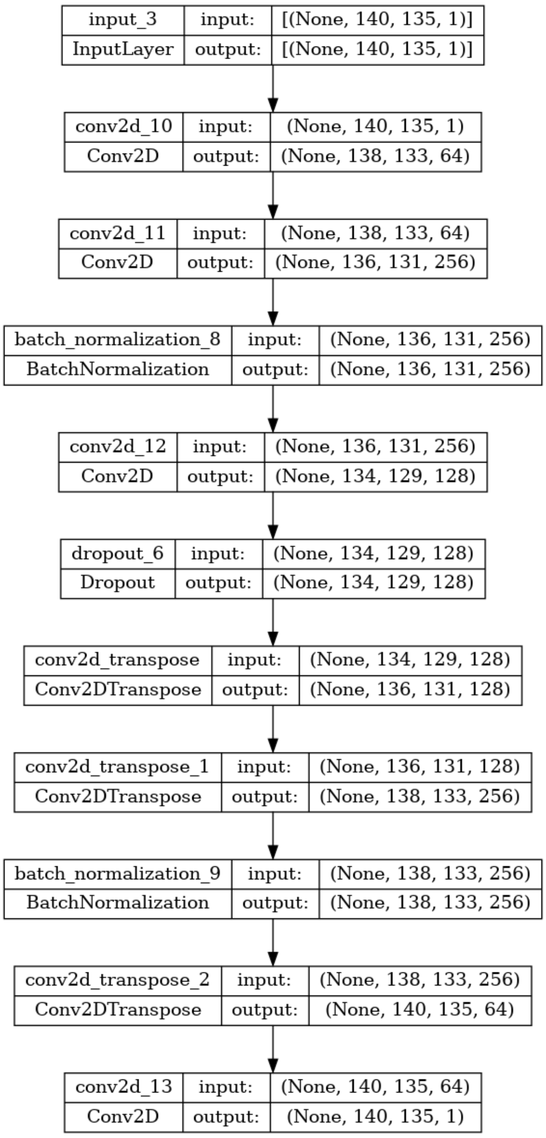
[3] K. N. Haque, M. A. Yousuf, and R. Rana, "Image denoising and restoration with cnn-lstm encoder decoder with direct attention." `https://arxiv.org/pdf/1801.05141.pdf`, 2018.

5 Appendix - Model Architecture



input_2 | input: | [(None, 420, 540, 1)]
InputLayer | output: | [(None, 420, 540, 1)]
conv2d_5 | input: | (None, 420, 540, 1)
Conv2D | output: | (None, 420, 540, 64)
max_pooling2d_2 | input: | (None, 420, 540, 64)
MaxPooling2D | output: | (None, 210, 270, 64)
batch_normalization_4 | input: | (None, 210, 270, 64)
BatchNormalization | output: | (None, 210, 270, 64)
dropout_3 | input: | (None, 210, 270, 64)
Dropout | output: | (None, 210, 270, 64)
conv2d_6 | input: | (None, 210, 270, 64)
Conv2D | output: | (None, 210, 270, 256)
max_pooling2d_3 | input: | (None, 210, 270, 256)
MaxPooling2D | output: | (None, 105, 135, 256)
batch_normalization_5 | input: | (None, 105, 135, 256)
BatchNormalization | output: | (None, 105, 135, 256)
dropout_4 | input: | (None, 105, 135, 256)
Dropout | output: | (None, 105, 135, 256)
conv2d_7 | input: | (None, 105, 135, 256)
Conv2D | output: | (None, 105, 135, 256)
up_sampling2d_2 | input: | (None, 105, 135, 256)
UpSampling2D | output: | (None, 210, 270, 256)
batch_normalization_6 | input: | (None, 210, 270, 256)
BatchNormalization | output: | (None, 210, 270, 256)
dropout_5 | input: | (None, 210, 270, 256)
Dropout | output: | (None, 210, 270, 256)
conv2d_8 | input: | (None, 210, 270, 256)
Conv2D | output: | (None, 210, 270, 64)
up_sampling2d_3 | input: | (None, 210, 270, 64)
UpSampling2D | output: | (None, 420, 540, 64)
batch_normalization_7 | input: | (None, 420, 540, 64)
BatchNormalization | output: | (None, 420, 540, 64)
conv2d_9 | input: | (None, 420, 540, 64)
Conv2D | output: | (None, 420, 540, 1)

Figure 1: CNN (without image crop)



input_3 | input: | [(None, 140, 135, 1)]
InputLayer | output: | [(None, 140, 135, 1)]
conv2d_10 | input: | (None, 140, 135, 1)
Conv2D | output: | (None, 138, 133, 64)
conv2d_11 | input: | (None, 138, 133, 64)
Conv2D | output: | (None, 136, 131, 256)
batch_normalization_8 | input: | (None, 136, 131, 256)
BatchNormalization | output: | (None, 136, 131, 256)
conv2d_12 | input: | (None, 136, 131, 256)
Conv2D | output: | (None, 134, 129, 128)
dropout_6 | input: | (None, 134, 129, 128)
Dropout | output: | (None, 134, 129, 128)
conv2d_transpose | input: | (None, 134, 129, 128)
Conv2DTranspose | output: | (None, 136, 131, 128)
conv2d_transpose_1 | input: | (None, 136, 131, 128)
Conv2DTranspose | output: | (None, 138, 133, 256)
batch_normalization_9 | input: | (None, 138, 133, 256)
BatchNormalization | output: | (None, 138, 133, 256)
conv2d_transpose_2 | input: | (None, 138, 133, 256)
Conv2DTranspose | output: | (None, 140, 135, 64)
conv2d_13 | input: | (None, 140, 135, 64)
Conv2D | output: | (None, 140, 135, 1)

Figure 2: CNN (with image crop)

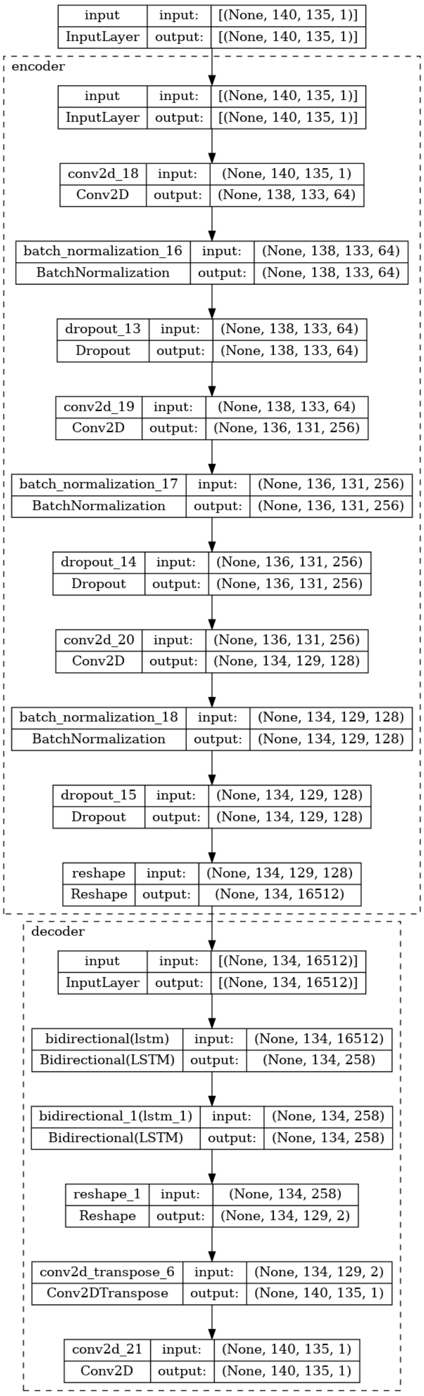

input input: [(None, 140, 135, 1)]
InputLayer output: [(None, 140, 135, 1)]
encoder
input input: [(None, 140, 135, 1)]
InputLayer output: [(None, 140, 135, 1)]
conv2d_18 input: (None, 140, 135, 1)
Conv2D output: (None, 138, 133, 64)
batch_normalization_16 input: (None, 138, 133, 64)
BatchNormalization output: (None, 138, 133, 64)
dropout_13 input: (None, 138, 133, 64)
Dropout output: (None, 138, 133, 64)
conv2d_19 input: (None, 138, 133, 64)
Conv2D output: (None, 136, 131, 256)
batch_normalization_17 input: (None, 136, 131, 256)
BatchNormalization output: (None, 136, 131, 256)
dropout_14 input: (None, 136, 131, 256)
Dropout output: (None, 136, 131, 256)
conv2d_20 input: (None, 136, 131, 256)
Conv2D output: (None, 134, 129, 128)
batch_normalization_18 input: (None, 134, 129, 128)
BatchNormalization output: (None, 134, 129, 128)
dropout_15 input: (None, 134, 129, 128)
Dropout output: (None, 134, 129, 128)
reshape input: (None, 134, 129, 128)
Reshape output: (None, 134, 16512)
decoder
input input: [(None, 134, 16512)]
InputLayer output: [(None, 134, 16512)]
bidirectional(lstm) input: (None, 134, 16512)
Bidirectional(LSTM) output: (None, 134, 258)
bidirectional_1(lstm_1) input: (None, 134, 258)
Bidirectional(LSTM) output: (None, 134, 258)
reshape_1 input: (None, 134, 258)
Reshape output: (None, 134, 129, 2)
conv2d_transpose_6 input: (None, 134, 129, 2)
Conv2DTranspose output: (None, 140, 135, 1)
conv2d_21 input: (None, 140, 135, 1)
Conv2D output: (None, 140, 135, 1)

Figure 3: CNN+LSTM

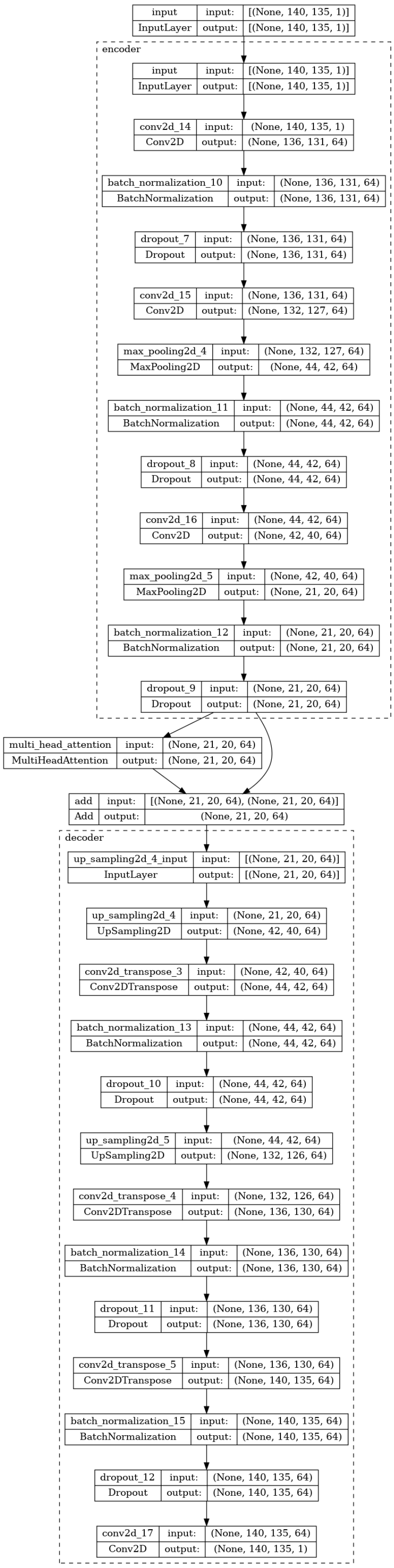

input input: [(None, 140, 135, 1)]
InputLayer output: [(None, 140, 135, 1)]
encoder
input input: [(None, 140, 135, 1)]
InputLayer output: [(None, 140, 135, 1)]
conv2d_14 input: (None, 140, 135, 1)
Conv2D output: (None, 136, 131, 64)
batch_normalization_10 input: (None, 136, 131, 64)
BatchNormalization output: (None, 136, 131, 64)
dropout_7 input: (None, 136, 131, 64)
Dropout output: (None, 136, 131, 64)
conv2d_15 input: (None, 136, 131, 64)
Conv2D output: (None, 132, 127, 64)
max_pooling2d_4 input: (None, 132, 127, 64)
MaxPooling2D output: (None, 44, 42, 64)
batch_normalization_11 input: (None, 44, 42, 64)
BatchNormalization output: (None, 44, 42, 64)
dropout_8 input: (None, 44, 42, 64)
Dropout output: (None, 44, 42, 64)
conv2d_16 input: (None, 44, 42, 64)
Conv2D output: (None, 42, 40, 64)
max_pooling2d_5 input: (None, 42, 40, 64)
MaxPooling2D output: (None, 21, 20, 64)
batch_normalization_12 input: (None, 21, 20, 64)
BatchNormalization output: (None, 21, 20, 64)
dropout_9 input: (None, 21, 20, 64)
Dropout output: (None, 21, 20, 64)
multi_head_attention input: (None, 21, 20, 64)
MultiHeadAttention output: (None, 21, 20, 64)
add input: [(None, 21, 20, 64), (None, 21, 20, 64)]
Add output: (None, 21, 20, 64)
decoder
up_sampling2d_4_input input: [(None, 21, 20, 64)]
InputLayer output: [(None, 21, 20, 64)]
up_sampling2d_4 input: (None, 21, 20, 64)
UpSampling2D output: (None, 42, 40, 64)
conv2d_transpose_3 input: (None, 42, 40, 64)
Conv2DTranspose output: (None, 44, 42, 64)
batch_normalization_13 input: (None, 44, 42, 64)
BatchNormalization output: (None, 44, 42, 64)
dropout_10 input: (None, 44, 42, 64)
Dropout output: (None, 44, 42, 64)
up_sampling2d_5 input: (None, 44, 42, 64)
UpSampling2D output: (None, 132, 126, 64)
conv2d_transpose_4 input: (None, 132, 126, 64)
Conv2DTranspose output: (None, 136, 130, 64)
batch_normalization_14 input: (None, 136, 130, 64)
BatchNormalization output: (None, 136, 130, 64)
dropout_11 input: (None, 136, 130, 64)
Dropout output: (None, 136, 130, 64)
conv2d_transpose_5 input: (None, 136, 130, 64)
Conv2DTranspose output: (None, 140, 135, 64)
batch_normalization_15 input: (None, 140, 135, 64)
BatchNormalization output: (None, 140, 135, 64)
dropout_12 input: (None, 140, 135, 64)
Dropout output: (None, 140, 135, 64)
conv2d_17 input: (None, 140, 135, 64)
Conv2D output: (None, 140, 135, 1)

Figure 4: CNN+Attention