Deep Learning Homework (Megajánlott jegyért)

Group: Zerocool

ISIC 2024 - Skin Cancer Detection with 3D-TBP

Team Members:

- Néder Brúnó (NQKZUX)
- Jenei Ákos (EBBUE7)
- Tasi Gergő (BRY27P)

1. Introduction

Deep learning in medical applications has an increasingly big role. Identifying skin cancer at an early stage can save lives. Our project aims to create a Convolutional Neural Network (CNN) to help identify skin cancer in patients using images of affected areas. We will utilize the ISIC 2024 Challenge dataset, provided by Kaggle, which contains a wide range of dermoscopic images (15x15 mm field-of-view cropped photos) for skin cancer diagnosis. The dataset also includes a lot of metadata along with images, such as where the picture is located on the patient's body and the results of different levels of lesion diagnosis. We aim to accurately predict whether the provided image (with metadata) is malignant or benign.

For more information about the dataset, you can visit the competition page on Kaggle: https://www.kaggle.com/competitions/isic-2024-challenge

2. Architecture Overview

2.1 Preprocessing

The data from Kaggle (ISIC 2024 Challenge dataset) first needed to be preprocessed. We have made some data cleaning changes to the database's metadata outside of the code:

- We removed columns that we felt were unnecessary or that would've ruined the learning capability of the model.
- In the age column, there was incomplete data (missing values). We replaced the empty fields with the average age of the patients. (58)
- Changed the raw text type of the photo's location to a one-hot-coded style, with 8 new columns.

2.2 Preprocessing with code

The next preprocessing steps were done with code.

Since there was a huge amount of (400k+) negative and 393 positive cases, we didn't have a good ratio to teach the model with. To solve this issue, we used a combination of two different approaches:

- First, we separated the cancerous and non-cancerous photos, so we could artificially change the ratio of malignant data points by loading only 3000 non-cancerous images (this is controlled by the images_to_load variable at the beginning of the code). This was done by dropping the rest of the non-cancerous metadata and concatenating these with the cancerous ones.
- We also used augmentation to help the model distinguish between the two classes. With the default setup (duplicate_cancerous_imgs = True, duplicate_non_cancerous_imgs = False) only the cancerous images (and metadata) are augmented. This process is later described in more detail.

To increase the learning accuracy of the model, we use Min-Max scaling (from sklearn.preprocessing) on numeric metadata (the not one-hot-coded values).

Since the metadata also included the names of the image files, we needed to drop these once the images were loaded. This is done inside a dataset generator, more on this later.

2.3 Splitting the data

We split the main dataset into train and test sets with a 20%-80% ratio and then split the training dataset into train and validation sets, again with a 20%-80% ratio.

We also used random shuffling at this stage, which was important, because the cancerous and non-cancerous data was concatenated and would come in order.

2.4 Dataset generators and data augmentation

Since we needed to work with a huge number of images, loading all of them into memory at the same time was infeasible. To solve this issue, we used datasets created from a generator function (with tf.data.Dataset.from_generator).

Inside the generator function we handled the loading and preprocessing of the images and augmenting both the images and the metadata. With the configuration variables, we can set whether we want to augment either the cancerous or both kinds of images.

We created a create_dataset function so that we could easily create the needed datasets (train, test, valid), that would generate data with the correct output signature that could be passed to the model directly. Here, we used another shuffling and created batches of 32.

2.4.1 Loading and preprocessing the images

We loaded the images based on the filenames from the metadata, after loading an image, we could drop this column because it wouldn't have helped the learning. After that, we resized the images to 125x125 pixels so we could work with them uniformly. The images are then normalized by dividing by 255. We also considered z-score normalization, but it didn't affect the result.

2.4.2 Image augmentation

If an image is augmented (if cancerous, for example), the image is duplicated num_augmented_versions (another config variable) times with an ImageDataGenerator. We use horizontal and vertical flipping and random normal noise.

2.4.3 Metadata augmentation

The metadata is also augmented with the augmented image. For this, we use the augment_variation hyperparam, which controls how much the values might be randomly changed. For example, if the augment_variation is 0.1, it multiplies data by a random number between 0.9 and 1.1. Since not all values might be augmented, we used a list of column names to exclude from augmentation ('isic_id', 'target', 'head_neck', 'torso_front', 'torso_back', 'right_leg', 'right_arm', 'left_leg', 'left_arm', 'unknown')

2.5 Combined CNN and Dense Network Model

We used a CNN to process the images and a dense network to deal with metadata. The CNN calculates an output for the image (a “feature vector”), which is then fed into the dense network along with the metadata.

2.5.1 CNN Branch

Input: Images of shape (125, 125, 3).

Layers:

- Convolutional layers with a configurable number of filters and kernels.
- MaxPooling layers for dimensionality reduction.
- Flatten layer to convert the spatial features into a vector.
- Dense layers for final embedding before combining.

2.5.2 Dense Branch:

Input: Additional (non-image) features.

Layers:

- Multiple Dense layers with ReLU activation.
- Final output layer with a sigmoid activation for binary classification.

2.5.3 Combined Output:

- The CNN output vector and the Dense branch input are concatenated.
- A series of Dense layers process the combined representation.
- A final sigmoid layer produces a probability between 0 and 1.

2.6 Hyperparameter Tuning

We used Keras Tuner to find the optimal hyperparameters:

CNN Hyperparameters:

- Number of convolutional layers.
- Number of neurons in each layer.
- Number of filters in convolutional layers.
- Number of units in the CNN's dense layer.

Dense Network Hyperparameters:

- Number of layers
- Number of units in each fully connected layer.

2.7 Training & Validation

Loss Function: binary_crossentropy

Optimizer: Adam (with default params)

Metrics: Accuracy

Callbacks:

- Early Stopping (monitoring validation accuracy).
- ModelCheckpoint (saving the best model).

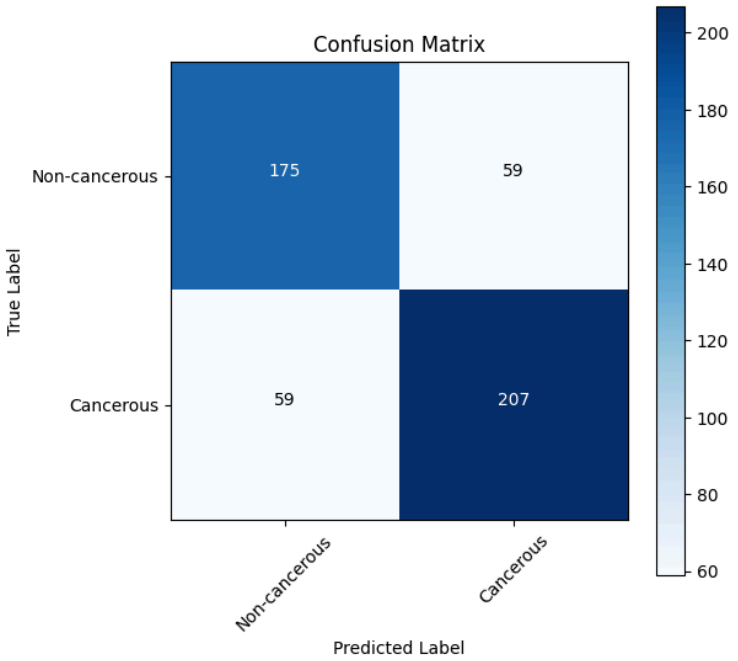
During training:

- Image data and additional features are provided in parallel.
- Validation data is used to monitor model performance and trigger early stopping if no improvements occur.

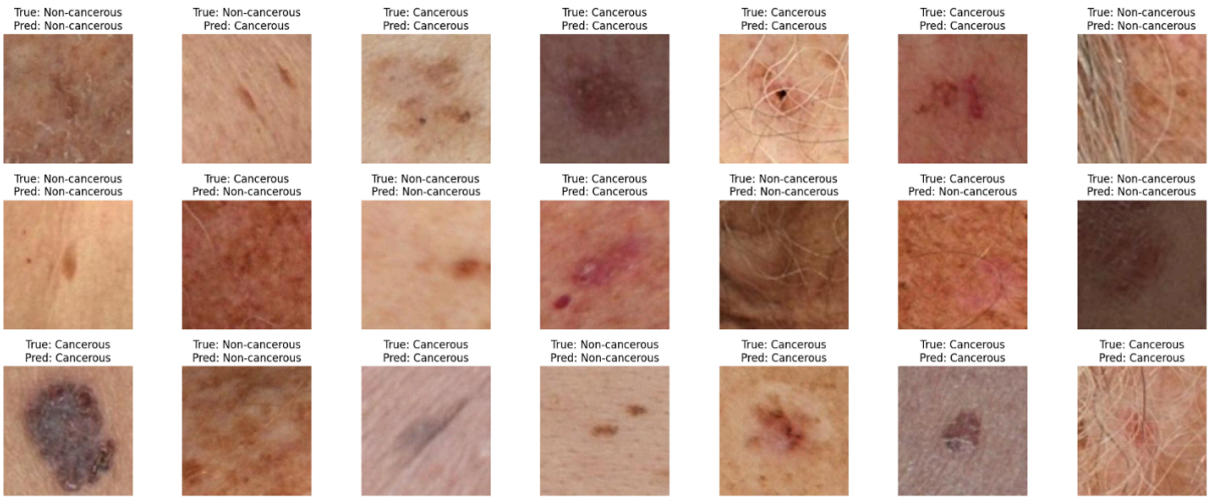
3. Results

The final model has an 81,62% test accuracy (and ~84% validation accuracy).

To check how our model performs, we created a confusion matrix: which shows the performance in terms of true positives, false positives, true negatives, and false negatives:



And finally, here are some demo predictions with images:



4. Conclusion

This project successfully implemented a hybrid CNN and dense network architecture to classify skin cancer using images and metadata. We were happy about the result, our goal was to reach around 80% accuracy, which we managed to do.

Possible improvements include a way to denoise or remove distracting parts of the image. For example, one of the TDKs this year was about removing hair and other markings from similar images, we could use that to improve detection.

5. Disclaimer

Usage of LLMs: We used LLMs (like ChatGPTor copilot) to help find and fix errors and bugs.