CNN algorithm for skin lesion classification

Abstract
Melanoma is a type of cutaneous malignancies and is known as the great mimicker in pathology. It can mimic other types of cutaneous lesions with variations in morphology remain largely unknown (Klebanov et al., 2019). Thus, the diagnostic accuracy of melanoma has been an ever-present challenge for pathologists. To solve this problem, a Convolutional neural network algorithm-based model with 12 layers was developed to distinguish melanoma and other skin lesions using Keras. Our model shows certain ability to identify malignant melanoma from non-malignant skin lesions and distinguish keratinocytic lesions from melanocytic ones. We also provide some insights for the improvement of our current model to improve classification accuracy, especially on the commonly existing overfitting problem. The using of larger datasets and more classical architecture, adding of call back functions provides directions for the future refinement as well.

Introduction

Melanoma is a fatal and fast-growing skin cancer around the world. It acts by affecting melanocytes which capable to cause skin to turn into black color and it can happen anywhere on human body. It is noticed that the mortality rate of melanoma can be reduced up to 90%, if early diagnosis is enabled. However, melanoma is difficult to recognize due to its similar appearances and symptoms as nevus and seborrheic keratosis. Thus, we are determined to build a convolutional neural network algorithm to distinguish melanoma and two types of benign skin lesions, which are nevus and seborrheic keratosis.

Based on a morphology expert generated ground-truth, a deep learning approach can benefit melanoma early detection with image analysis. The algorithm can be further combined with high-resolution imaging techniques such as dermoscopy to improve diagnosis accuracy. This can not only reduce the manpower and time required for melanoma diagnosis; on the other hand, potential patients or patients under treatment can perform the investigation themselves conveniently using a dermatoscopy-attached smartphone, which greatly benefits patient care.

In this report, we will briefly describe how we generate a deep learning model using convolutional neural network (CNN), and its performance on predicting images for different skin lesions. The image dataset we used is from ISIC Challenge 2017- Skin Lesion Analysis Towards Melanoma Detection – Part 3: Lesion Classification (Codella et al. 2018) (Download link: https://challenge.kitware.com/#challenge/583f126bcad3a51cc66c8d9a). It has three subdirectories, respectively train directory, validation directories, and test directory. Each subdirectory contains images for melanoma, nevus and seborrheic keratosis (Fig.3).

Algorithm

1. Pseudocode & algorithm workflow

```
#data preprocessing using imageDataGenerator
#model building
def build_model():
model=model.sequentials()
#the number of Con2d_layer, Maxpooling_layer depends on real tests
model.add(Conv2d_layer,activation_function)
model.add(Maxpooling_layer)
model.add(Dense_layer,activation_function)
model.add(Dropout_layer)
model.add(Dense_layer,activation_function='softmax')#categorical
model.compile()#optimizer, ini_learning_rate
return model
model=build_model()
#training the model
History=model.fit(train_data, validation_data, epoch_num)
#history stores all data in the procedure
#draw leraning curves for training and validation against epochs
plot(History['train_acc'],History['val_acc'],epochs)
plot(History['train_loss'], History['val_loss'],epochs)
#model prediction
Pred=Model.predict(test_data)
def prediction_probability_result(Predition) #store probability distribution in csv
prediction_probability_result(Pred)
#confusion matrix
def plot_confusion_matrix(Pred,Ground_truth)
#ROC curve plot and AUC
def plot_roc(Pred,Ground_truth)
def auc_score(Pred,Ground_truth)
```

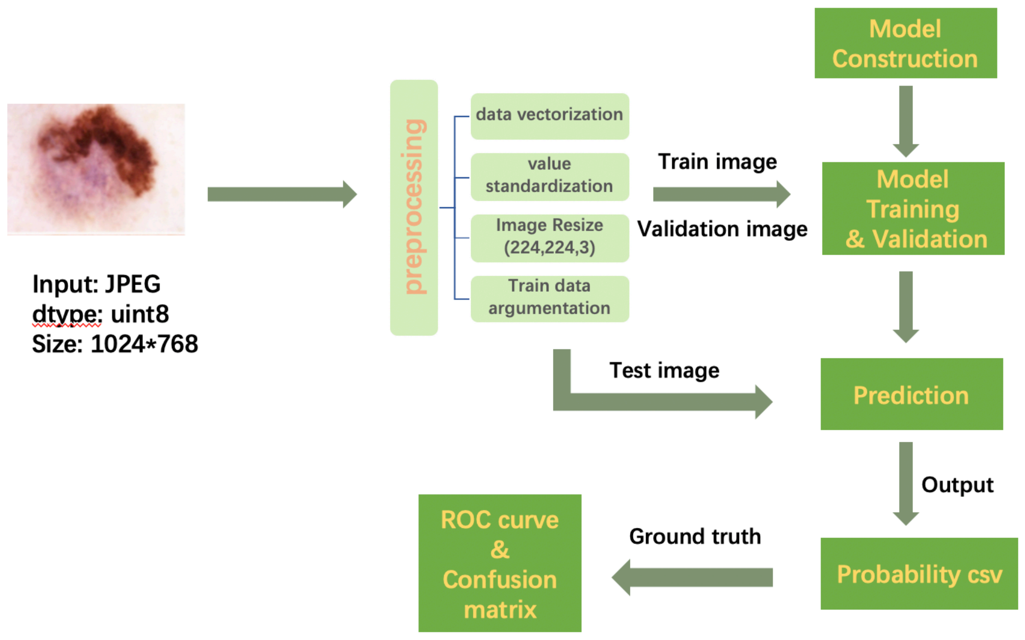
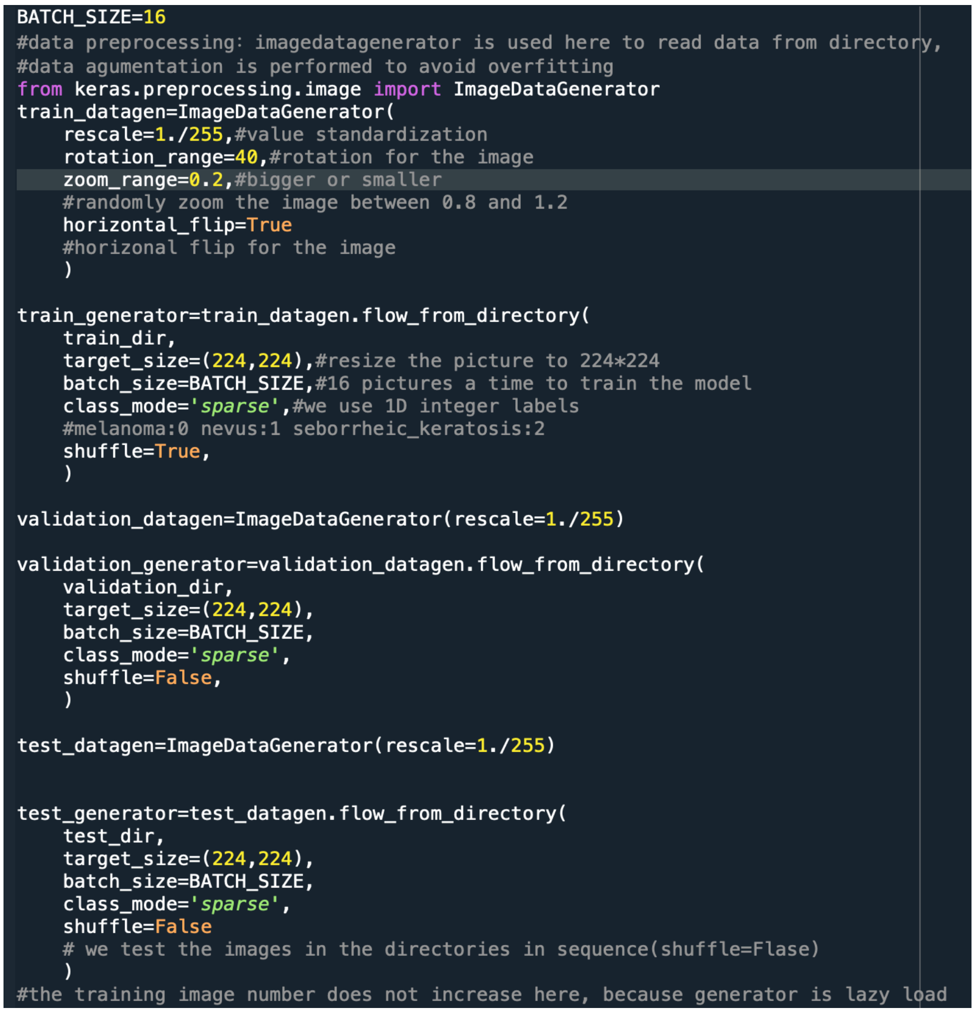


Figure 1: Mini project flow chart. The input JPEG images first undergo preprocessing, then train and validation images are used to train a CNN model, while test images are used for evaluation. The prediction of test images would be output in a csv and further used for visualization.

2. Data preprocessing

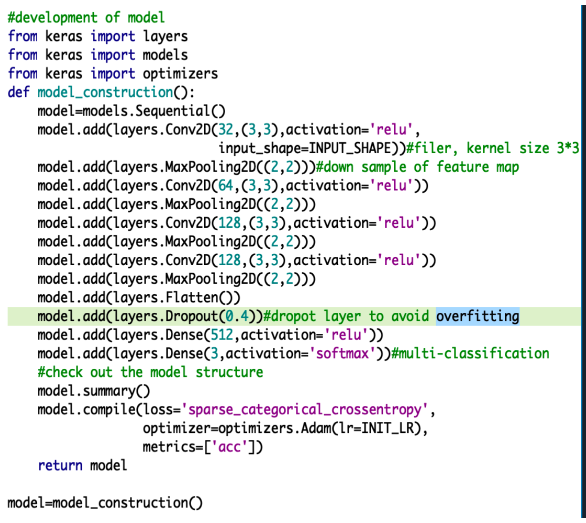
The original dataset downloaded from the ISCI Challenge 2017 provided a train dataset of a total 2000 images, among which 1300+ are nevus. Considering that the imbalance distribution of training dataset would cause reduced learning efficiency, we randomly selected 20% nevus pictures to generate a new nevus training data. All images were read in and decoded to RGB from JPEG automatically by modules in Keras.preprocessing.images package. Value standardization was required to deal with heterogeneous data. As RGB images are encoded with integers ranging from 0 to 255 to represent color intensity, we divided them by 255, thus all input values would lie between 0 and 1. By doing so, we can avoid the extreme gradient update caused by features with great difference value, therefore speed up the convergence of the network. Then, we performed data vectorization for our input images and labels, turning their data types to float32 tensors which can be received by convolutional neural network. Also, it was noticed that the training dataset size is relatively small, which would result in the poor performance of model. With little training data, an over-constraint model is more likely to underfit the data, due to lack of hypothesis space. Otherwise, an under-constraint model tends to overfit the training data. To minimize the effect, data argumentation was carried out to expand the dataset size. During training, the original images in each minibatch would undergo horizontal flip, zoom, and rotation transformations. We did not perform the shear transformation on training images since it might disturb feature recognition. To notice, only the processed images rather than the original ones would be used to train the CNN. All images used were resized to (224,224). For detailed implementation, ImageDataGenerator class in Keras.preprocessing.images package was recruited for all data preprocessing steps mentioned above. By using ImageDataGenerator class, image batches were yielded without having to load the entire dataset, eliminating the memory requirement.



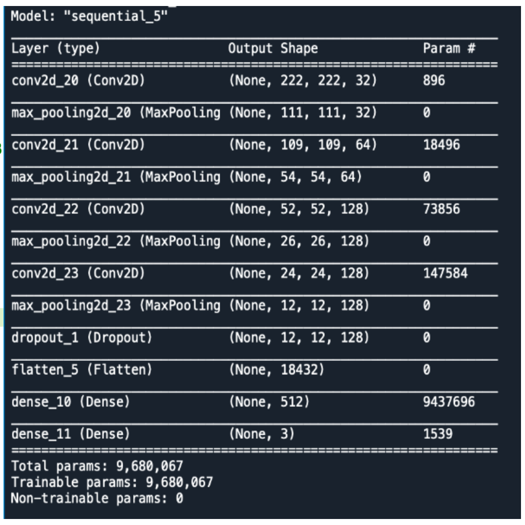
```
BATCH_SIZE=16
#data preprocessing: imagedatagenerator is used here to read data from directory,
#data agumentation is performed to avoid overfitting
from keras.preprocessing.image import ImageDataGenerator
train_datagen=ImageDataGenerator(
    rescale=1./255,#value standardization
    rotation_range=40,#rotation for the image
    zoom_range=0.2,#bigger or smaller
    #randomly zoom the image between 0.8 and 1.2
    horizontal_flip=True
    #horizonal flip for the image
    )

train_generator=train_datagen.flow_from_directory(
    train_dir,
    target_size=(224,224),#resize the picture to 224*224
    batch_size=BATCH_SIZE,#16 pictures a time to train the model
    class_mode='sparse',#we use 1D integer labels
    #melanoma:0 nevus:1 seborrheic_keratosis:2
    shuffle=True,
    )

validation_datagen=ImageDataGenerator(rescale=1./255)

validation_generator=validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(224,224),
    batch_size=BATCH_SIZE,
    class_mode='sparse',
    shuffle=False,
    )

test_datagen=ImageDataGenerator(rescale=1./255)


test_generator=test_datagen.flow_from_directory(
    test_dir,
    target_size=(224,224),
    batch_size=BATCH_SIZE,
    class_mode='sparse',
    shuffle=False
    # we test the images in the directories in sequence(shuffle=Flase)
    )
#the training image number does not increase here, because generator is lazy load
```

3. Model construction, fitting & prediction

Compared with its predecessors, a prominent advantage of convolutional neural network is that it can automatically detect important image features without any human supervision. Based on this characteristic, we developed a CNN with 12 layers to classify medical images. The selection of hyperparameters (the number of hidden layers and units) was under guidance of the model performances on training and validation data, with which we gradually increased the number of hidden layers to provide the model with a bigger memorization capacity to better learn the reflection between training samples and targets. Also, we enlarge the number of hidden units within a layer to avoid underfitting. However, an overfitting problem appeared. To solve this, a dropout layer was added, by randomly dropping units along connections from the neural network to prevent units from over co-adaptations, and the exact dropout rate is selected based on practice. To build the model from bottom to top, we used model and layers modules provided by Keras to accomplish the task. At the beginning of the model architecture, a Conv2D layer designed with 32 hidden units was used to accept the reshaped RGB image input using layers. conv2D function. It created a convolution kernel of 3*3 size to convolved with the layer input to produce a tensor of outputs and 'ReLu' (Rectified Linear Units) activation function was used to enlarge hypothesis space. The reason for choosing a kernel size of 3*3 rather than 5*5 is that a small kernel could better observe delicate features. Followed by the convolution layer was a max pooling layer that down-sampled feature maps. It divided the output of convolution layer, feature map into small grids sized 2*2. The maximum value from each grid was collected to compile a reduced image matrix. By inserting max pooling layers between convolution layers, the number of parameters and calculations in network were effectively reduced, thus contributing to the control of overfitting. We repeated the feature extraction layers, that is the combination of convolution, activation, and maxpooling layes, for three more times. In this way, the window could cover a higher proportion of the original input, enabling the learning of spatial hierarchies of patterns. Then, a flatten layer was used to convert the output of features extraction layers into one-dimensional data with layers.Flatten() function. A dropout layer was added to ease overfitting via layers.Dropout() function. Next, a fully connected layer was created by layers.Dense() functionr to transform data dimensions for the feasibility of linearly classification. 'ReLu' function was used to activate the layer. The final layer dense layer with 3 units which activated by 'softmax' function, acted as a classier. Thus, for each input sample, the network would give out its probability distribution on each category with an overall probability equaled to 1.

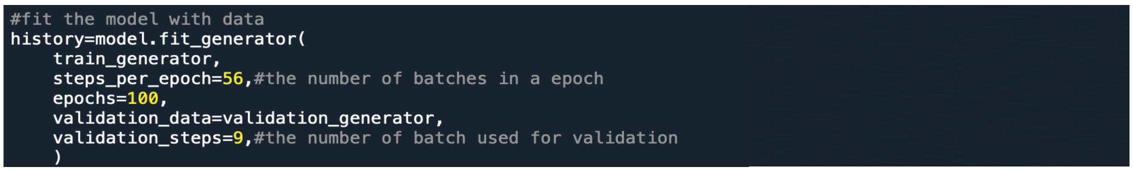


```
#development of model
from keras import layers
from keras import models
from keras import optimizers
def model_construction():
    model=models.Sequential()
    model.add(layers.Conv2D(32,(3,3),activation='relu',
                            input_shape=INPUT_SHAPE))#filer, kernel size 3*3
    model.add(layers.MaxPooling2D((2,2)))#down sample of feature map
    model.add(layers.Conv2D(64,(3,3),activation='relu'))
    model.add(layers.MaxPooling2D((2,2)))
    model.add(layers.Conv2D(128,(3,3),activation='relu'))
    model.add(layers.MaxPooling2D((2,2)))
    model.add(layers.Conv2D(128,(3,3),activation='relu'))
    model.add(layers.MaxPooling2D((2,2)))
    model.add(layers.Flatten())
    model.add(layers.Dropout(0.4))#dropot layer to avoid overfitting
    model.add(layers.Dense(512,activation='relu'))
    model.add(layers.Dense(3,activation='softmax'))#multi-classification
    #check out the model structure
    model.summary()
    model.compile(loss='sparse_categorical_crossentropy',
                  optimizer=optimizers.Adam(lr=INIT_LR),
                  metrics=['acc'])
    return model

model=model_construction()
```



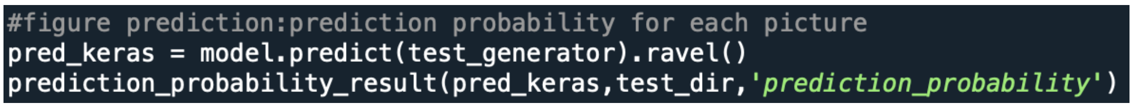
```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_20 (Conv2D)           (None, 222, 222, 32)      896
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 111, 111, 32)      0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 109, 109, 64)      18496
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 54, 54, 64)        0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 52, 52, 128)       73856
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 26, 26, 128)       0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 24, 24, 128)       147584
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 12, 12, 128)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 128)       0
_________________________________________________________________
flatten_5 (Flatten)          (None, 18432)             0
_________________________________________________________________
dense_10 (Dense)             (None, 512)               9437696
_________________________________________________________________
dense_11 (Dense)             (None, 3)                 1539
=================================================================
Total params: 9,680,067
Trainable params: 9,680,067
Non-trainable params: 0
```

When coming to model compilation, model.compile() function was used to set optimizer, loss function. As we aimed to perform multi-classification and our labels were coded with integer, ‘sparse_categorical_crossentrophy’ was selected as loss function to measure the distance between true value and our prediction. Besides, Adam optimizer was selected, and the initial learning rate was set to 3e-4 according to actual practices. To improve the reusability of model, our model building and compilation were written in a model_construction() function. model.fit_generator() function was called to train our model, returning a dictionary object (‘history’) which contained all the data during the training procedure.



```
#fit the model with data
history=model.fit_generator(
    train_generator,
    steps_per_epoch=56,#the number of batches in a epoch
    epochs=100,
    validation_data=validation_generator,
    validation_steps=9,#the number of batch used for validation
    )
```

To visualize the training procedure, accuracy and loss curve for validation and training were plotted against epochs (Fig.2). After fitting the model, model.predict() function was recruited to generate output predictions for the test images yielded in test generator. The perdition probability for each sample would be collected and returned in a csv file by a self-designed function called prediction_probability_result().



```
#figure prediction:prediction probability for each picture
pred_keras = model.predict(test_generator).ravel()
prediction_probability_result(pred_keras,test_dir,'prediction_probability')
```

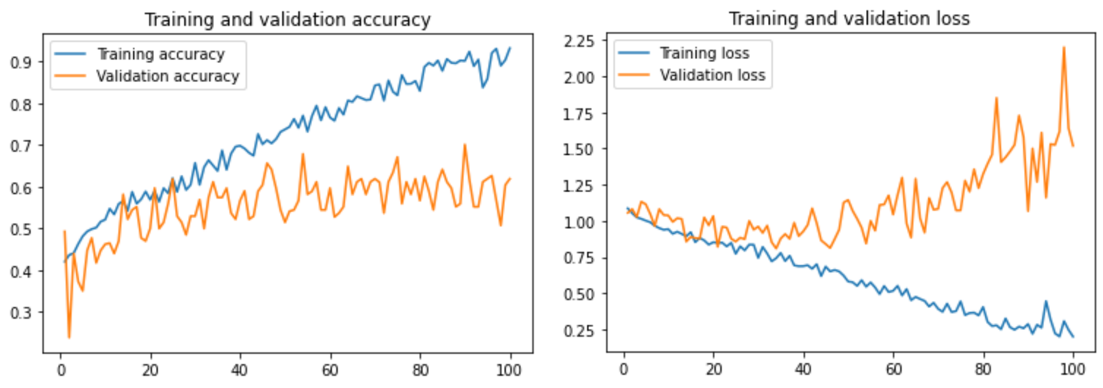


Figure 2: Learning curve plots. x: epochs; y: accuracy or loss. After 100 epochs of training, our model reached a max training accuracy of 0.932 and a minimal training loss of 0.200. The max validation accuracy is 0.702, and min validation loss is 0.809.

Result

1. Data overview

Our skin lesion image dataset consists of 1652 JPEG images, each of which has 1024*768 pixels in dimension. Among the 1652 images, 902 are in the training directory, 150 are in the validation directory and the final 600 in the test directory. Each image in the three directories was grouped into the corresponding class folder (“melanoma”, “nevus” and “seborrheic_keratosis”) according to ground truth (Fig.3).

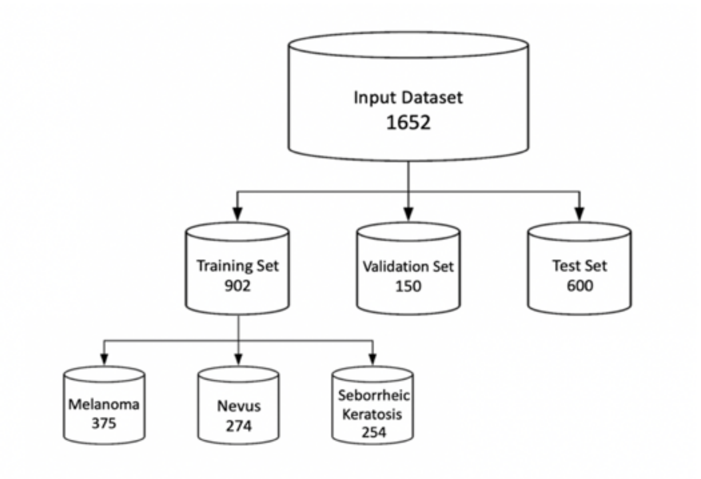
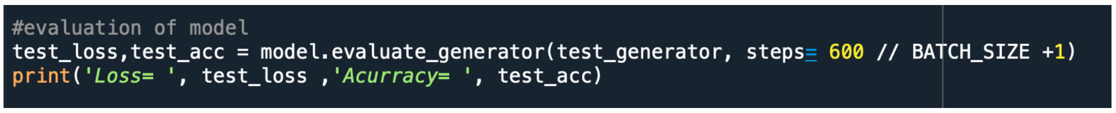


Figure 3: Input dataset structure. The input dataset we used are splited into training, validation, and testing sets. Under each subset, the images are grouped into melanoma, nevus, or seborrheic keratosis folders according to ground truth.

2. Model evaluation & output

After generating the 12-layer CNN model, we trained the model for 100 epochs. The running time for each epoch is around 140-160s, However, due to the availability of GPU usage on server, the running time greatly varied. The virtual memory usage on remote GPU server is 163688 kb.

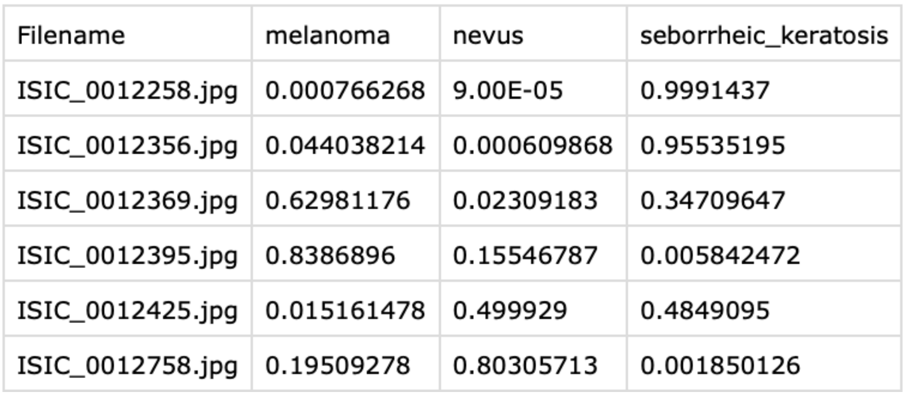
We first provide an overview of the model accuracy using evaluate_generator from Keras to print out the loss and accuracy of our model:



```
#evaluation of model
test_loss,test_acc = model.evaluate_generator(test_generator, steps= 600 // BATCH_SIZE +1)
print('Loss= ', test_loss ,'Acurracy= ', test_acc)
```

[Loss = 2.3159565381209055 Acurracy = 0.53]

Then, we define a function to output the prediction result as a csv file. Our output csv contains 4 columns, which are filenames and predicted probabilities for the three classes of skin lesion for images by model, respectively (Fig.4). We also allow users to use our pre-trained model to test their own images, and give the predicted probability in the output csv. This is accomplished by a command line executable file ("prediction_probability.py") using argparse package (see supplymentary materials for implementation details).



Filename	melanoma	nevus	seborrheic_keratosis
ISIC_0012258.jpg	0.000766268	9.00E-05	0.9991437
ISIC_0012356.jpg	0.044038214	0.000609868	0.95535195
ISIC_0012369.jpg	0.62981176	0.02309183	0.34709647
ISIC_0012395.jpg	0.8386896	0.15546787	0.005842472
ISIC_0012425.jpg	0.015161478	0.499929	0.4849095
ISIC_0012758.jpg	0.19509278	0.80305713	0.001850126

Figure 4: The head rows of output csv file "prediction_probability.csv". Columns: filenames, probability of the test image being melanoma, nevus, or seborrheic keratosis. For example, for the first test image ISIC_0012258.jpg, our model output a 0.9991437 confidence that this image belongs to seborrheic keratosis class.

3. Accuracy measurement

For model evaluation, we further visualized our model performance using confusion matrix and ROC – AUC.

Confusion matrix

Confusion matrix is a specific method used to visualize the performance of the algorithm, usually in supervised learning. The columns in the confusion matrix are the predicted results for each type of lesions obtained from our model, while the rows represent the true attribution category of the images. Figure 5 shows an exact and a normalized confusion matrix of our prediction result.

For each test image in the output csv, the index of the biggest possibility value is retrieved as its predicted category result. The actual category is retrieved from the ground truth. We used confusion_matrix class imported from sklearn.metrics to compare predicted category with true category and mapped the result into a cnf_matrix array. The highest prediction rate for each category is all on the diagonal, showing that the model can predict the skin lesion class correctly to some extent.

```
#confusion matrix
from sklearn.metrics import confusion_matrix
#obtain predicted probabilities
scores = model.predict_generator(
    generator=test_generator,
    steps=600 // BATCH_SIZE + 1)
#retrieve the largest probability index as predicted result
#combine with true catogory from test_generator to get cnf_matrix array
cnf_matrix = confusion_matrix(test_generator.classes, list(map(lambda x: np.argmax(x), scores)))
plot_confusion_matrix(cnf_matrix, classes=class_names, title='Confusion matrix')
```

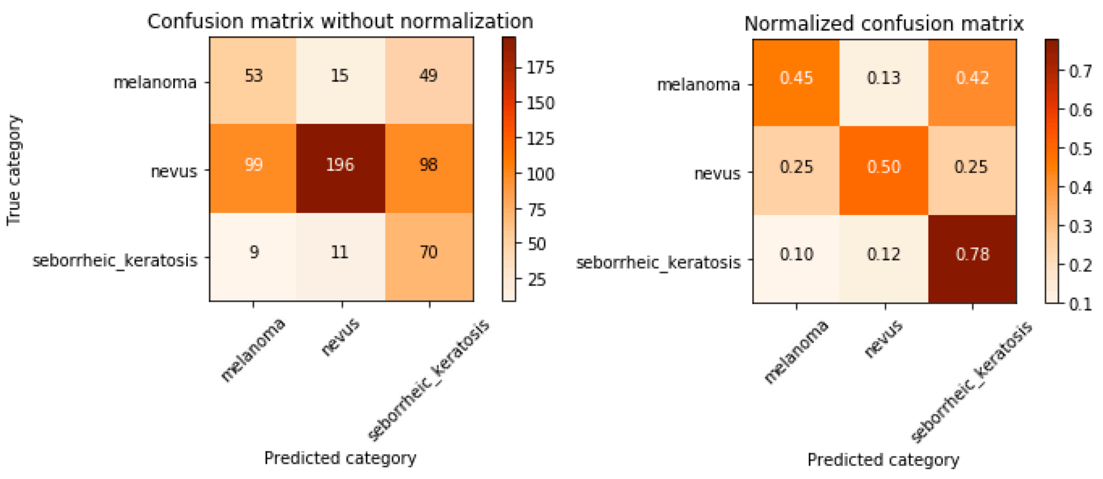


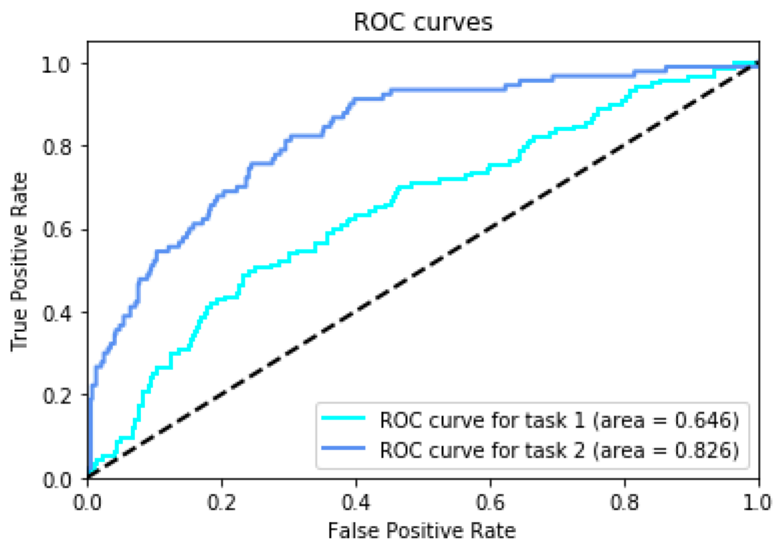
Figure 5: Normalized and unnormalized confusion matrix. If we use greatest probability as the prediction result criterion, the true positive rate for predicting melanoma, nevus and seborrheic keratosis are respectively 0.45, 0.50 and 0.78. Darker color patch in the matrix represents a larger number/higher probability

ROC and AUC

Two Receiver operating characteristic (ROC) curve was generated to visualize our model performance. Area Under the Curve, AUC is used as a score to reflect the corresponding performance. Usually, the ROC curve is used to illustrate the ability of a binary classifier under changing thresholds from 0 to 1. When there are three or more classes, ROC curve can be extended by one-vs-rest approach. Inspired by the ISIC 2017 challenge, we binarized the outcome, plotted ROC curves and calculated AUC scores respectively for the following two tasks (Fig.6).

- **Task 1: Malignant vs non-malignant.** This can gauge the ability of the model to distinguish between the malignant melanoma and the other two benign skin lesions.
- **Task 2: Keratinocytic vs melanocytic.** Unlike melanoma and nevus which are melanocytic, seborrheic keratosis is derived from keratinocytes. We want to test the performance of our model on distinuishing between melanocytic and keratinocytic skin tumor.

The ROC curves showed that our CNN worked better on classifying between keratinocytic and melanocytic skin tumor then distinguish melanoma from benign skin lesions, which is consistent with the confusion matrix. We also see this kind of pattern on other group's model performance in ISIC challenge 2017 leaderboard.



```
Task 1 Score: 0.646
Task 2 Score: 0.826
Mean Score: 0.736
```

Figure 6: ROC curve and ROC AUC value for each task. For melanoma and benign skin lesions distinguishment, our model reached a score of 0.646. For Keratinocytic vs melanocytic task our model reached a score of 0.826. The mean AUC score of the two tasks is 0.736. Python code reference for ROC curve and AUC: https://github.com/udacity/dermatologist-ai/blob/master/get_results.py.

Discussion

1. Overfitting

We found that during the first few attempts, the loss of the validation data would first decrease, but then it starts to oscillate and then even rises. However, the validation accuracy after several epochs remains the same with high training accuracy. With further reading, it is inferred to be a feature of train data overfitting. The large number of hidden layers in deep neural networks enables them to learn very related and complicated features. Thus, if the training data is small, the network may recognize sampling noises as complicated relationships, which will not be presented in validation and test data (Srivastava et al. 2014).

By ignoring some of the feature detectors (making some of the hidden layer nodes value equal to zero) in each training batch, over-fitting can be significantly reduced. This method is called dropout, which can make the model more general because it will not rely too much on some local features.

In our final model, we used dropout layer imported form keras after the flatten layer as suggest in "Deep Learning with Python" (Chollet, 2017). This effectively improves our model's max validation accuracy by 13.6%. Although this is accompanied by a slowdown in the fitting speed of the training data, this is normal and also proved the dropout solution had played a role. We also tried VGG-like design, which add several Dropout(0.2) layers after pooling layers, but we did not see very significant improvement.

Besides already adopted solutions, a common way to solve overfitting was to minimize the structure of network. However, this is not suitable for our situation. As the differences between the three subtypes are minor, reducing the network structure brings about worries that the feature differences between each type cannot be well learned due to the compacted hypothesis space. It is also found on git-hub that more complex network structures such as VGG16/19 or Resnet, can more effectively use the data and have a better classification performance (some of the participants also used extra train data from trusted sources to train more complex models). Because CNN model performance is reliant on training images, enlarged training dataset that contains more various feature of melanoma are thought to helpful on supervised melanoma classification. A recent study used an unsupervised K-means algorithm to perform morphological sub-groupings on melanoma images. (Klebanov et al., 2019) With a larger collection of morphologically diverse melanoma datasets, in combination with a deeper network, the classification accuracy can be further improved.

2. Call back function

Another improvement to make is the setting of call back functions. During the training procedure, it is noticed that sometimes the validation accuracy is stuck somewhere until reaching the end of epochs. By using call back function, early stopping of the training procedure is enabled, meanwhile the best model obtained in training is saved. This effectively shortens the training time, avoids the excessive redundancy running and saves the computational resources, such as memory usage. Also, the use of call back function can help with the dynamic modulation of parameters. For example, reduce the learning rate when the validation loss stops to improve.

3. Training data imbalance

In data preprocessing, we attempted to enhance learning efficiency by resizing the training dataset. However, the processed training dataset is still imbalanced though less severe than before, in which melanoma images occupying a larger proportion than nevus and seborrheic keratosis. This may induce a higher possibility for the network to take others type of skin lesions as melanoma, which may increase the false positive rate. But we decide to keep this train data composition. False-positive or taking others as melanoma, is considered to be less serious than false-negative which means mistaken melanoma as moderate ones, which might lead to late treatment and a higher fatality rate.

Later we realized that what we use is not an ideal solution to deal with train data imbalance. In real cases, there will always be imbalanced training data, and reducing the train samples will omit a lot of useful resources. For this, an extra data collection is a reliable method. If no more data can be retrieved, oversampling for minor classes can be carried out to reach the balance instead.

References

Chollet F., (2017), 'Deep learning with Python', Chapter 5, page 113, Manning Publication

Codella, N. C. F. *et al.* (2018) 'Skin lesion analysis toward melanoma detection: A challenge at the 2017 International symposium on biomedical imaging (ISBI), hosted by the international skin imaging collaboration (ISIC)', in *Proceedings - International Symposium on Biomedical Imaging*. doi: 10.1109/ISBI.2018.8363547.

He, T. , Zhang, Z. , Zhang, H. , Zhang, Z. , Xie, J. , & Li, M. (2019), 'Bag of tricks for image classification with convolutional neural networks', *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June, pp. 558–567.

Klebanov, N., Gunasekera, N. S., Lin, W. M., Hawryluk, E. B., Miller, D. M., Reddy, B. Y., Christman, M. P., Beaulieu, D., Rajadurai, S., Duncan, L. M., Sober, A. J., & Tsao, H. (2019). Clinical spectrum of cutaneous melanoma morphology. *Journal of the American Academy of Dermatology*, 80(1), 178–188.e3.

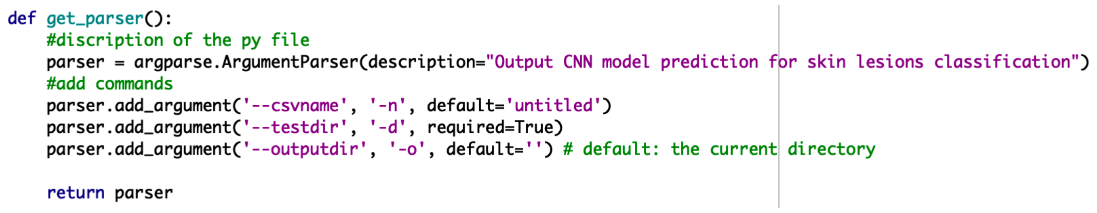
Matsunaga, K., A. Hamada., A. Minagawa, H.Koga, (2017), 'Image classification of melanoma, nevus and seborrheic keratosis by deep neural network ensemble', *arXiv*, pp. 2–5.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014), Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research,15*(1), 1929-1958.

Supplementary materials

Python	3.7.4	numpy	1.15.4
Tensorflow	1.15.4	sklearn	0.23.0
Keras-Application	1.0.8	Matplotlib	3.3.3
Keras-preprocessing	1.1.2		

Table1. Table for versions of tools and packages used



```
def get_parser():
    #discription of the py file
    parser = argparse.ArgumentParser(description="Output CNN model prediction for skin lesions classification")
    #add commands
    parser.add_argument('--csvname', '-n', default='untitled')
    parser.add_argument('--testdir', '-d', required=True)
    parser.add_argument('--outputdir', '-o', default='') # default: the current directory

    return parser
```

Figure 1. Detailed information of execute command line executable prediction_probability.py file:

Command parameters: -n, output csv filrname
-d, user provided test directory
-o, output path

* To use our model for image classification, you need to have a ‘test’ folder containing your images under your provided directory.

implementation using command line: python3 prediction_probability.py -n your/output/filename -d your/test/dir -o your/output/dir