CLASSIFICATION ON SKIN DISEASE IMAGE USING TRANSFER LEARNING.

Ng Zhen Xiong

FCI Multimedia University

1181302692@student.mmu.edu.my

Imran Uzair

FCI Multimedia University

1171102458@student.mmu.edu.my

ABSTRACT

The task of the project is to classify different types of skin disease images into the respective classes. The dataset consists of 27k of common human skin disease images that are labelled in 10 classes. The goal is to utilize the TensorFlow platform and perform machine learning on the dataset and perform classification. Main approach is by using transfer learning on different models provided by the TensorFlow Keras API. The deep learning models used in this project are EfficientNetB2, XceptionNet, VGG19, InceptionV3 and ResNet50 with the validation accuracy of 78%, 81%, 78%, 74% and 82% respectively.

Index Terms— Machine learning, TensorFlow, Keras, Skin disease

1. INTRODUCTION

Skin disease is commonly diagnosed on human skin and it comes in different types. Different types of skin disease have different patterns and may occur on different parts of human skin. Deep learning can help in diagnosing the type of skin disease of an image quickly based on the patterns if it is well trained. In this project, we provide the deep learning models with a skin disease dataset with 27k images and 10 classes, namely: Eczema, Melanoma, Atopic Dermatitis, Basal Cell Carcinoma(BCC), Melanocytic Nevi (NV), Benign Keratosis-like Lesions (BKL), Psoriasis pictures Lichen Planus and related diseases, Seborrheic Keratoses and other Benign Tumors, Tinea Ringworm Candidiasis and other Fungal Infections,Warts Molluscum and other Viral Infections. The table below displays the breakdown of the images into the 10 different classes. We can observe that Melanocytic Nevi has a considerably larger amount of samples at 7970 images, while Atopic Dermatitis suffers from having the least amount at 1207 images to be exact. The rest of the classes however, lie around 1500-3000 images. This is important to take note of since the model might start to develop a bias towards images that are more represented, this is because there are more images for the network to find said features.

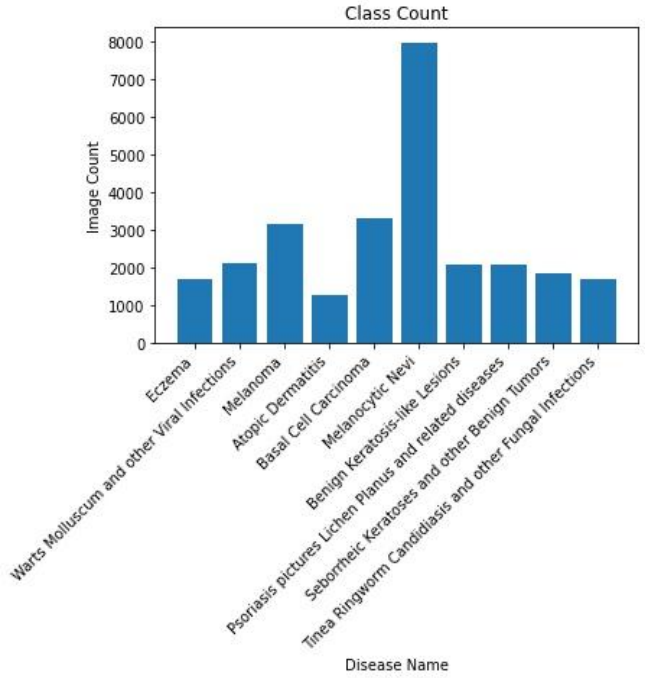

Figure 1: Skin disease image count of each class

Training a classification model from scratch is expensive on computing resources and requires a lot of time. Instead of creating it from scratch, we use transfer learning on our problem. Transfer learning consists of taking features learned on one problem, and leveraging them on a new, similar problem. In our case, we take the model that has trained on ImageNet weight and we add a new classification layer for our problem.

2. BACKGROUND

Medical imagery has been on the rise, especially with the prominence of deep learning within the last few years. Back in 2014, a paper titled, "Medical image classification with convolutional network" [1], explored the thought of classifying images of lung patches with interstitial lung disease using CNNs. Instead of taking a transfer learning approach, they opted to create their own network from scratch. In their paper, they described that they were able to outperform three SVM-based approaches.

Another field of medicine where CNN is looking to be applied is cancer diagnosis [2]. Manual detection of cancer cells is both tiresome and prone to human error. With the introduction of image classification through a CNN, they developed a model to take in pathological images and classify them into 'Benign' and 'Malignant'. The dataset that they used contained images of different zoom levels totalling up to 7909 images. Using their model they achieved a prediction accuracy of 99.86%.

3. APPROACH

We use TensorFlow and the Keras API provided to perform transfer learning on our problem. The first step is to perform pre-processing on the skin disease image dataset. First we split the data into training dataset and validation dataset. The validation split is 0.2 and we resize the image based on the recommendation on TensorFlow documentation for each model, for example the width and height for ResNet50 should not be less than 32. Next we pre- process the image input by calling the built-in preprocess input function in TensorFlow, this will help preprocesses a tensor or Numpy array encoding a batch of images.

It is suggested that when training on smaller dataset, we should use data augmentation to create more data to reduce the overfitting of a model. We use the augmentation provided by TensorFlow such as RandomRotation, Random Translation, RandomFlip and RandomContrast to tweak our dataset and create more data for training. We implement it using Keras Sequential function.

The next step is to initiate a base model and load pre-trained weight to it. The base models we use here are EfficientNetB2, XceptionNet, VGG19, InceptionV3 and ResNet50 and the weight loaded is ImageNet. We will first freeze all the layers in the base model and train a new top model to classify our dataset. The top model consists of different layers such as Dense layer, Flatten layer and Dropout layer depending on the need of different base models. Now that we have a top layer to learn about our problem and won't affect the base model, this step is called feature extraction. After extracting the feature of our problem, we can now fine tune the model by unfreezing a few layers of the base model. With that we can have more parameters to learn on our problem and help achieve better results.

Note that learning rate is important when doing fine tuning to the model. We implemented callbacks to lower down the learning rate when the validation accuracy stops improving. This callback is created using the ReduceLROnPlateau from Keras API.

Last, we evaluate the performance of the models by printing out the training accuracy, training loss, validation accuracy and validation loss. Using these metrics we can then plot a graph to monitor the performance and tweak the parameters to retrain the model to get better performance.

4. EXPERIMENT

The experiment is conducted using various models shown at the below paragraph. The parameter that was used to train the model and the performance result will be shown using figure and graph.

Dataset

The dataset used for this project is a skin disease image dataset with 27 images in 10 classes. The skin disease dataset can be found on kaggle: https://www.kaggle.com/ismailpromus/skin-diseases-image-dataset.

Feature Extraction

All the models use the same parameter for extracting. The model for the feature is just a short one with only 5 epochs for the model to learn on the target data. The details of the parameters are shown in the table.

Parameter	Value
Epochs	5
Batch size	64
Optimizer	Adam
Learning Rate	1e-3
Loss	Sparse Categorical Cross Entropy

Fine Tune Model

All the models share the same parameter to fine tune the model. Unfreezing the last 30 layers of the base model seems to give good results for every model plus we are running out of time for experimenting with different parameters and layers. The table shows the parameter use for fine tune models:

Parameter	Value
Epochs	30
Batch size	64
Optimizer	Adam
Learning Rate	1e-4
Loss	Sparse Categorical Cross Entropy
Callback	ReduceLROnPlateau
Layer unfreeze	30

Callback Function

A ReduceLROnPlateau callback function is created to auto adjust the learning rate of fine tune model to monitor the validation accuracy:

Parameter	Value
Monitor	val_accuracy
Patience	3
Verbose	1
Factor	0.5
min_lr	0.000001
cooldown	2

4.1 ResNet50

4.1.1 Top model layers

Layers	Value	Activation
GlobalAveragePooling2D	nan	nan
BatchNormalization	nan	nan
Dropout	0.5	nan
Dense	512	ReLU
BatchNormalization	nan	nan
Dropout	0.5	nan
Dense	10	Softmax

4.1.2 Feature extraction performance

```
Epoch 1/5
725/725 [==============================] - 92s 118ms/step - loss: 1.3919 - accuracy: 0.5497 - val_loss: 0.9126 - val_accuracy: 0.6613
Epoch 2/5
725/725 [==============================] - 87s 120ms/step - loss: 1.0326 - accuracy: 0.6162 - val_loss: 0.8507 - val_accuracy: 0.6783
Epoch 3/5
725/725 [==============================] - 90s 123ms/step - loss: 0.9809 - accuracy: 0.6305 - val_loss: 0.8280 - val_accuracy: 0.6842
Epoch 4/5
725/725 [==============================] - 89s 123ms/step - loss: 0.9370 - accuracy: 0.6472 - val_loss: 0.8041 - val_accuracy: 0.6864
Epoch 5/5
725/725 [==============================] - 89s 123ms/step - loss: 0.9196 - accuracy: 0.6522 - val_loss: 0.8407 - val_accuracy: 0.6796
```

4.1.3 Performance of last 5 epochs after fine tune

```
Epoch 25/30
725/725 [==============================] - 102s 140ms/step - loss: 0.3441 - accuracy: 0.8754 - val_loss: 0.5672 - val_accuracy: 0.8190 - lr: 2.5000e-05
Epoch 26/30
725/725 [==============================] - 102s 140ms/step - loss: 0.3357 - accuracy: 0.8783 - val_loss: 0.5578 - val_accuracy: 0.8230 - lr: 2.5000e-05
Epoch 27/30
```

```
725/725 [==============================] - 102s
140ms/step - loss: 0.3342 - accuracy: 0.8794 - val_loss:
0.5875 - val_accuracy: 0.8186 - lr: 2.5000e-05
Epoch 28/30
725/725 [==============================] - 102s
140ms/step - loss: 0.3124 - accuracy: 0.8884 - val_loss:
0.5651 - val_accuracy: 0.8184 - lr: 2.5000e-05
Epoch 29/30
724/725 [==============================>.] - ETA:
0s - loss: 0.3078 - accuracy: 0.8884
Epoch 00029: ReduceLROnPlateau reducing learning rate
to 1.249999968422344e-05.
725/725 [==============================] - 102s
140ms/step - loss: 0.3078 - accuracy: 0.8884 - val_loss:
0.5919 - val_accuracy: 0.8199 - lr: 2.5000e-05
Epoch 30/30
725/725 [==============================] - 101s
140ms/step - loss: 0.2856 - accuracy: 0.8999 - val_loss:
0.5769 - val_accuracy: 0.8236 - lr: 1.2500e-05
```

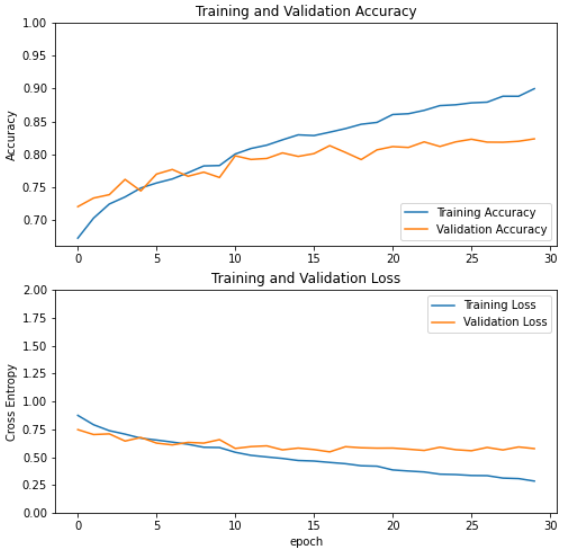

Figure 2: The accuracy and loss performance of ResNet50

4.2 **EfficientNetB2**

4.2.1 Top model layers

Layers	Value	Activation
Flatten	nan	nan
Dense	256	ReLU
Dropout	0.5	nan
Dense	10	Softmax

4.2.2 Feature extraction performance

```
Epoch 1/5
340/340 [==============================] - 157s
406ms/step - loss: 2.2173 - accuracy: 0.3985 - val_loss:
1.3122 - val_accuracy: 0.4853
Epoch 2/5
340/340 [==============================] - 132s
386ms/step - loss: 1.4696 - accuracy: 0.4526 - val_loss:
1.1568 - val_accuracy: 0.5215
Epoch 3/5
340/340 [==============================] - 190s
556ms/step - loss: 1.3627 - accuracy: 0.4765 - val_loss:
1.1216 - val_accuracy: 0.5267
Epoch 4/5
340/340 [==============================] - 181s
530ms/step - loss: 1.3185 - accuracy: 0.4933 - val_loss:
1.0856 - val_accuracy: 0.5589
Epoch 5/5
340/340 [==============================] - 166s
486ms/step - loss: 1.2902 - accuracy: 0.4973 - val_loss:
1.0668 - val_accuracy: 0.5948
```

4.2.3 Performance of last 5 epochs after fine tune

```
Epoch 25/30
340/340 [==============================] - 123s
361ms/step - loss: 0.4957 - accuracy: 0.8143 - val_loss:
0.7697 - val_accuracy: 0.7713 - lr: 1.0000e-04
Epoch 26/30
340/340 [==============================] - 123s
361ms/step - loss: 0.4781 - accuracy: 0.8198 - val_loss:
0.7977 - val_accuracy: 0.7715 - lr: 1.0000e-04
Epoch 27/30
340/340 [==============================] - 122s
358ms/step - loss: 0.4595 - accuracy: 0.8294 - val_loss:
0.7523 - val_accuracy: 0.7744 - lr: 1.0000e-04
Epoch 28/30
340/340 [==============================] - 123s
361ms/step - loss: 0.4434 - accuracy: 0.8358 - val_loss:
0.8190 - val_accuracy: 0.7737 - lr: 1.0000e-04
Epoch 29/30
340/340 [==============================] - 123s
360ms/step - loss: 0.4224 - accuracy: 0.8445 - val_loss:
0.8498 - val_accuracy: 0.7735 - lr: 1.0000e-04
Epoch 30/30
340/340 [==============================] - 124s
363ms/step - loss: 0.4068 - accuracy: 0.8508 - val_loss:
0.8258 - val_accuracy: 0.7773 - lr: 1.0000e-04
```

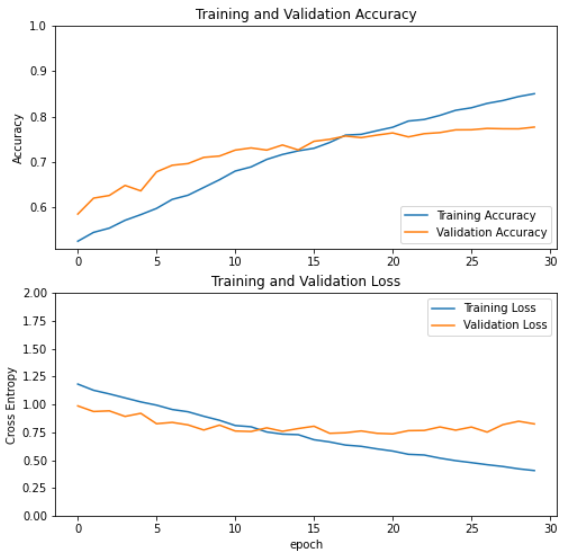

Figure 3: The accuracy and loss performance of EfficientNetB2

4.3 XceptionNet

4.3.1 Top model layers

Layers	Value	Activation
Flatten	nan	nan
Dense	512	ReLU
Dropout	0.7	nan
Dense	10	Softmax

4.3.2 Feature extraction performance

```
Epoch 1/5
725/725 [==============================] - 61s 73ms/step - loss: 2.0553 - accuracy: 0.3694 - val_loss: 1.4120 - val_accuracy: 0.4477
Epoch 2/5
725/725 [==============================] - 54s 74ms/step - loss: 1.5783 - accuracy: 0.4004 - val_loss: 1.2897 - val_accuracy: 0.4692
Epoch 3/5
725/725 [==============================] - 53s 73ms/step - loss: 1.5433 - accuracy: 0.4045 - val_loss: 1.2564 - val_accuracy: 0.4726
Epoch 4/5
725/725 [==============================] - 55s 75ms/step - loss: 1.5376 - accuracy: 0.4138 - val_loss: 1.2114 - val_accuracy: 0.4961
Epoch 5/5
725/725 [==============================] - 56s 77ms/step - loss: 1.5352 - accuracy: 0.4206 - val_loss: 1.2755 - val_accuracy: 0.4858
```

4.3.3 Performance of last 5 epochs after fine tune

```
Epoch 25/30
725/725 [==============================] - 74s 101ms/step - loss: 0.1671 - accuracy: 0.9420 - val_loss: 0.8942 - val_accuracy: 0.7959 - lr: 5.0000e-05
Epoch 26/30
725/725 [==============================] - 74s 101ms/step - loss: 0.1589 - accuracy: 0.9445 - val_loss: 0.9009 - val_accuracy: 0.8074 - lr: 5.0000e-05
Epoch 27/30
725/725 [==============================] - 74s 101ms/step - loss: 0.1583 - accuracy: 0.9459 - val_loss: 0.8378 - val_accuracy: 0.8048 - lr: 5.0000e-05
Epoch 28/30
725/725 [==============================] - 74s 102ms/step - loss: 0.1460 - accuracy: 0.9482 - val_loss: 0.9238 - val_accuracy: 0.8068 - lr: 5.0000e-05
Epoch 29/30
724/725 [==============================>.] - ETA: 0s - loss: 0.1432 - accuracy: 0.9489
Epoch 00029: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-05.
725/725 [==============================] - 74s 102ms/step - loss: 0.1433 - accuracy: 0.9489 - val_loss: 0.9156 - val_accuracy: 0.8041 - lr: 5.0000e-05
Epoch 30/30
725/725 [==============================] - 74s 101ms/step - loss: 0.1191 - accuracy: 0.9583 - val_loss: 0.8914 - val_accuracy: 0.8149 - lr: 2.5000e-05
```

Figure 4: The accuracy and loss performance of XceptionNet

4.4. VGG19

4.4.1 Top model layers

Layers	Value	Activation
GlobalAveragePooling2D	nan	nan
BatchNormalization	nan	nan
Dropout	0.5	nan
Dense	512	ReLU
BatchNormalization	nan	nan
Dropout	0.5	nan
Dense	10	Softmax

4.4.2 Feature extraction performance

```
Epoch 1/5
725/725 [==============================] - 158s 200ms/step - loss: 1.6462 - accuracy: 0.4769 - val_loss: 1.0554 - val_accuracy: 0.6083
Epoch 2/5
725/725 [==============================] - 124s 171ms/step - loss: 1.2679 - accuracy: 0.5332 - val_loss: 1.0253 - val_accuracy: 0.6101
Epoch 3/5
725/725 [==============================] - 95s 130ms/step - loss: 1.2017 - accuracy: 0.5496 - val_loss: 0.9765 - val_accuracy: 0.6256
Epoch 4/5
725/725 [==============================] - 128s 176ms/step - loss: 1.1772 - accuracy: 0.5591 - val_loss: 0.9622 - val_accuracy: 0.6389
Epoch 5/5
725/725 [==============================] - 109s 150ms/step - loss: 1.1633 - accuracy: 0.5595 - val_loss: 0.9684 - val_accuracy: 0.6304
```

4.4.3 Performance of last 5 epochs after fine tune

```
Epoch 25/30
725/725 [==============================] - 239s 329ms/step - loss: 0.4423 - accuracy: 0.8410 - val_loss: 0.6592 - val_accuracy: 0.7720 - lr: 2.5000e-05
Epoch 26/30
725/725 [==============================] - 229s 316ms/step - loss: 0.4294 - accuracy: 0.8492 - val_loss: 0.7126 - val_accuracy: 0.7628 - lr: 2.5000e-05
Epoch 27/30
725/725 [==============================] - 230s 317ms/step - loss: 0.4113 - accuracy: 0.8531 - val_loss: 0.6878 - val_accuracy: 0.7656 - lr: 2.5000e-05
Epoch 28/30
725/725 [==============================] - ETA: 0s - loss: 0.3962 - accuracy: 0.8584
Epoch 00028: ReduceLROnPlateau reducing learning rate to 1.249999968422344e-05.
725/725 [==============================] - 216s 298ms/step - loss: 0.3962 - accuracy: 0.8584 - val_loss: 0.6795 - val_accuracy: 0.7663 - lr: 2.5000e-05
Epoch 29/30
725/725 [==============================] - 235s 324ms/step - loss: 0.3675 - accuracy: 0.8706 - val_loss: 0.6509 - val_accuracy: 0.7779 - lr: 1.2500e-05
Epoch 30/30
725/725 [==============================] - 216s 297ms/step - loss: 0.3581 - accuracy: 0.8735 - val_loss: 0.6508 - val_accuracy: 0.7814 - lr: 1.2500e-05
```

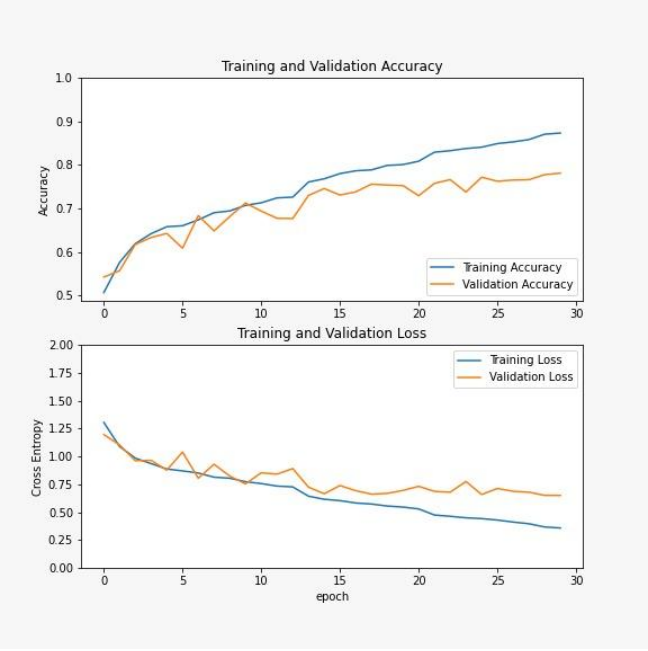

Figure 5: Accuracy and loss performance for Vgg19

4.5 Inception

4.5.1 Top model layers

Layers	Value	Activation
GlobalAveragePooling2D	nan	nan
BatchNormalization	nan	nan
Dropout	0.5	nan
Dense	512	ReLU
BatchNormalization	nan	nan
Dropout	0.5	nan
Dense	10	Softmax

4.4.2 Feature extraction performance

```
Epoch 1/5
725/725 [==============================] - 65s 78ms/step - loss: 1.5910 - accuracy: 0.4809 - val_loss: 1.1158 - val_accuracy: 0.5759
Epoch 2/5
725/725 [==============================] - 55s 76ms/step - loss: 1.2159 - accuracy: 0.5418 - val_loss: 1.0489 - val_accuracy: 0.5904
Epoch 3/5
725/725 [==============================] - 51s 70ms/step - loss: 1.1716 - accuracy: 0.5538 - val_loss: 1.0345 - val_accuracy: 0.5926
Epoch 4/5
725/725 [==============================] - 51s 70ms/step - loss: 1.1429 - accuracy: 0.5609 - val_loss: 1.0231 - val_accuracy: 0.6022
Epoch 5/5
725/725 [==============================] - 51s 70ms/step - loss: 1.1244 - accuracy: 0.5694 - val_loss: 1.0122 - val_accuracy: 0.6059
```

4.4.3 Performance of last 5 epochs after fine tune

```
Epoch 25/30
725/725 [==============================] - 60s 83ms/step - loss: 0.4877 - accuracy: 0.8238 - val_loss: 0.7732 - val_accuracy: 0.7361 - lr: 5.0000e-05
Epoch 26/30
725/725 [==============================] - 60s 82ms/step - loss: 0.4725 - accuracy: 0.8277 - val_loss: 0.7804 - val_accuracy: 0.7374 - lr: 5.0000e-05
Epoch 27/30
724/725 [============================>.] - ETA: 0s - loss: 0.4731 - accuracy: 0.8297
Epoch 00027: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-05.
725/725 [==============================] - 60s 82ms/step - loss: 0.4734 - accuracy: 0.8297 - val_loss: 0.7851 - val_accuracy: 0.7376 - lr: 5.0000e-05
Epoch 28/30
725/725 [==============================] - 59s 81ms/step - loss: 0.4500 - accuracy: 0.8373 - val_loss: 0.7724 - val_accuracy: 0.7401 - lr: 2.5000e-05
Epoch 29/30
725/725 [==============================] - 60s 82ms/step - loss: 0.4448 - accuracy: 0.8400 - val_loss: 0.7727 - val_accuracy: 0.7418 - lr: 2.5000e-05
Epoch 30/30
725/725 [==============================] - 59s 81ms/step - loss: 0.4394 - accuracy: 0.8404 - val_loss: 0.7699 - val_accuracy: 0.7396 - lr: 2.5000e-05
```

Figure 6: Accuracy and loss performance for InceptionV3

5. CONCLUSION

In this project we applied transfer learning using several different models to to classify 10 different diseases. They were trained and tested on a dataset of over 27000 images. We concluded that the best performing model was ResNet50, scoring an accuracy of 82%. We were able to achieve this level of accuracy using only a basic level of data augmentation and by adding a few layers at the end of the model. With further work and tuning, it is more than possible that the applications of CNNs in real world use could be expanded and improved upon, especially where expert analysis and classification plays an important role within that field. However, with all that being said, a key area that could impede the growth of CNNs as a classification method is the dataset that you wish to train the model with. There are various variables to take into account when looking for an appropriate dataset. For one, the size of the dataset plays a huge factor in the training of a network. Without the necessary amount of training, the network may not be able to learn the features that are required to be able to classify said images into their appropriate classes. Secondly, the environment that the images are being taken in also affects the performance of the model. For example, having a dataset that has a noisier background might suffer in terms of performance, since the model has to adapt to the irrelevant features that are present within the images. This might be useful later on if you would like to be able to classify images that are captured in a more natural setting. Choosing the right dataset is a tight-rope that you will have to balance on, depending on the requirements that are being demanded.

4.6 Performance Comparison

The table below shows the performance of each model on every metric after 30 epochs.

	Training Accuracy	Training Loss	Validation Accuracy	Validation Loss
Efficient NetB2	0.8508	0.4068	0.7773	0.8258
ResNet50	0.8999	0.2856	0.8236	0.5769
Inception V3	0.8404	0.4394	0.7396	0.7699
VGG19	0.8735	0.3581	0.7814	0.6508
Xception	0.9583	0.1191	0.8149	0.8914

11. REFERENCES

[1] Qing Li, Weidong Cai, Xiaogang Wang, Yun Zhou, David Dagan Feng and Mei Chen, "Medical image classification with convolutional neural network", Control Automation Robotics & Vision (ICARCV) 2014 13th International Conference on, pp. 844-848, 2014.

[2] Dabeer S., Khan M.M., Islam S., et al., “Cancer diagnosis in histopathological image: CNN based approach”, Inform. Med. Unlocked (2019)

[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] Chollet, F., & others. (2015). Keras. GitHub. Retrieved from https://github.com/fchollet/keras