Classifying Of Human Kidney Cells

John Ross Munn — 6646083

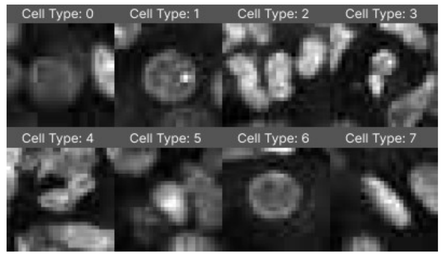
Introduction

The Objective

The aim of this project is to classify the cell type of an image with a 65% accuracy minimum using the most efficient machine learning techniques applicable. The model will be making use of this data by training on the provided dataset then running predictions on the test dataset and comparing the results on Kaggle to give a final accuracy for the model.

The Data

For this project, the dataset contains 28x28 (2D) grayscale images of human kidney cortex cells. These images are classified as 8 types of cells in this dataset. The dataset consists of 150,000 training images with their cell type identified in the attached CSV file and 50,000 test images for testing final model accuracy.



(Images of each cell type from the dataset.)

Research

When researching for this project, I looked for the most efficient method for classifying low-resolution cell images and found a variety of methods that could be implemented on this dataset.

The most similar project I found was a cancer cell classification journal article that made use of pre-trained models and transfer learning. This technique used Keras EfficientNet B0-B7 pre-trained models to use transfer learning to train their model achieving a maximum accuracy of 87.91% (Ali et al., 2022). The most useful tool I found in this article was the EfficientNet library as it provides multiple pre-trained models to test on your dataset and I would use this on my first implementation. However the dataset has a large effect on the efficiency of these models and unbalanced data can produce results with a lower accuracy than a convolution neural network that has not been trained via transfer learning (Ali et al., 2022).

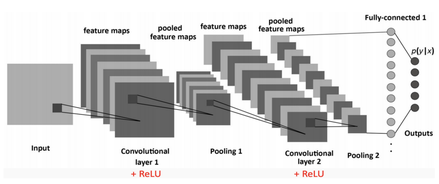
Upon finding this article I found more information on the specific Keras EfficientNet models and what I would need to utilize them for training my own model. I found many important sections of information of how to augment my data for each pre-trained model as well as a handbook of how to fine tune these models later in training (Team, n.d.)

EfficientNet B0 to B7

`EfficientNetB0` function

```
tf.keras.applications.EfficientNetB0(
    include_top=True,
    weights="imagenet",
    input_tensor=None,
    input_shape=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
    **kwargs
)
```

The last method that was relevant to the project was making a CNN (Convolutional neural network) as well as the layers that would need to be implemented so the transfer learning models had the most efficient layers and that if I implemented a CNN with no transfer learning I be able to affectively use layers. I found a detailed analysis of convolution and the filtering methods/augmentation applied to the images, along with the hyper parameters that would be most relevant when adjusting the model (Stewart, 2019).



Analysis — Transfer Learning

Research

In my research I found that keeping a consistent model between pre-trained model testing would help eliminate the models that wouldnt be as applicable for the dataset. Although I will just be testing the Keras library models I found there are a large quantity of other pre-trained models that I could also test however I decided against it as the Keras library covers a wide range of image training and would likely be a good indication on whether this method can produce precise results (Team, n.d.).

S.No	Pre-trained Models
1	VGG-16
2	DenseNet169
3	InceptionV3
4	Inception ResNet
5	ResNet50
6	AlexNet

(Other pre-trained models options from (Shamila Ebenezer et al., 2022))

Challenges

When implementing the EfficientNet it became clear that the dataset images were to small even for the smallest pre-trained model input size of (32x32), this is an issue as it causes the images to become padded and compressed which can result in a lower accuracy (Team, n.d.). The other challenge is that the project train dataset is unbalanced which means that some classifications will receive less training than others as there is less training data which in transfer learning can cause issues with accuracy.

```
Total images: 150000

Label 0: 48073          (32.05%)
Label 1: 7072           (4.71%)
Label 2: 5333           (3.56%)
Label 3: 13964          (9.31%)
Label 4: 10696          (7.13%)
Label 5: 7049           (4.70%)
Label 6: 35607          (23.74%)
Label 7: 22198          (14.80%)
```

(The unbalanced data from the training set)

Description Of Approach

With this method of classification I tested each EfficientNet pre-trained model with the same model layers to keep consistency between testing, with this method I could find the best EfficientNet model to use for training and make alterations to get the highest possible accuracy.

Before I began training the data had to be split into folders via csv using "shutil", then for the submission csv I used the Pandas DataFrame to construct the results by rounding to the highest prediction average

```
with open(filename, "r") as csvfile:
    datareader = csv.reader(csvfile)
    for row in datareader:
        if row[0] != "Id":
            shutil.move('train/' + row[0] + ".jpg", 'train/' + row[1])
```

(Shutil image separation)

```
from pandas import DataFrame
predictions_data = model.predict(test, use_multiprocessing=True)

predictions = []
for x in predictions_data:
    if x.argmax() == 0:
        predictions.append("0")
    elif x.argmax() == 1:
        predictions.append("1")
    elif x.argmax() == 2:
        predictions.append("2")
    elif x.argmax() == 3:
        predictions.append("3")
    elif x.argmax() == 4:
        predictions.append("4")
    elif x.argmax() == 5:
        predictions.append("5")
    elif x.argmax() == 6:
        predictions.append("6")
    elif x.argmax() == 7:
        predictions.append("7")

DataFrame(data={'Cell Type': predictions}).to_csv('predictions.csv', index_label='Id')
```

CSV generation with DataFrame

Implementation — Transfer Learning

Description

For this final implementation EfficientNet B3 was the best pre-trained model from my testing, I used the model in the figure below to train it using Dense layers with Dropout to keep a steady learning rate. I used B3 for my final implementation and testing doing 30 Epochs the maximum accuracy I was able to achieve with this approach was 48.54%.

Model

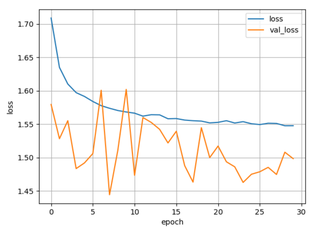
```
x = Flatten()(last_output)
x = Dropout(0.2)(x)
x = Dense(32, activation='relu')(x)
x = Dense(64, activation='relu')(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.1)(x)
x = Dense(64, activation='relu')(x)
output = Dense(8, activation='softmax')(x)

# Define optimizer, learning rate and loss function
model = Model(pre_train_model.input, output)
model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['acc'])
```

Results

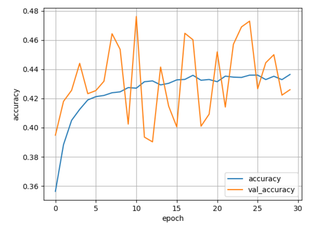
EfficientNet Model	Precision
B0	32.30%
B1	28%
B2	43.67%
B3	48.54%
B4	42.97%
B5	36.36%
B6	38.75%
B7	41.84%

Parameters

Input Shape	75
Batch Size	32
Data Split	80/20



For this model, the loss followed a normal downward trend however the val_loss had rapid changes each epoch likely due to the learning issues caused by the input size limitations.

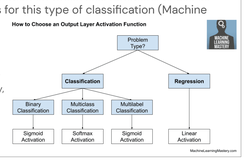


The same trend can be seen on this plot of data, the accuracy followed a normal trend but the val_accuracy rapidly changed after each epoch improving rapid then performing poorly after due to the same input issue

Analysis — CNN

Analysis— New Approach

After my first approach I learnt that this dataset was not ideal for transfer learning, with unbalanced data, low resolution images and large input sizes although it is possible and pre-trained models other than the keras library like "ResNet or InceptionV3" (Shamila Ebenezer et al., 2022) could result in a higher accuracy a convolution neural network would prove to be a better option with the quantity of data in the dataset (Convolutional Neural Network (CNN, 2019). Upon further research I found an applicable model from a similar dataset and some highly detailed information on which loss functions to use as well as activation functions for this type of classification (Machine Learning Mastery, 2016).



(Output layer options for classification (Machine Learning Mastery, 2016).)

Challenges

The main challenge I encountered was overfitting , when training for longer epochs the train data would begin to increase in accuracy but the validation data would plateau. This overfitting meant that images were over training on the same data resulting in accuracy constantly increasing. However I found a fix to this issue in the Keras library using image augmentation to artificially create more data, the amount of training data could effectively be doubled. The way this function works is it randomly flips images in the training set each batch so the training data is unique and random in longer epochs resulting in less overfitting, (TensorFlow, n.d.) and saw an overall accuracy increase of 3% after implementing this function!

(Example of random flip data augmentation)

Description Of Approach

For this next approach I implemented a CNN model with no transfer learning used, I got this model from the TensorFlow tutorial page but I altered some layers with the information I had learnt from completing the COM2028 labs.

This was because there was no Dropout to remove badly performing models or Dense layers which would be necessary for a increasing performance using a deeply connected layer. Lastly the batch sizes for each layer was to small for training with this much data so they were increased and made in increments of 2.

The data augmentation remained the same with the same validation split of 80/20 as well as the data sorting methods with the exception of the random flip function to decrease overfitting and increasing the batch size to 500 as the dataset is very large. I did experiment and alter layers through testing and the best model is what is being presented in the implementation.

Implementation — CNN

Description

The first layers are Conv2D which is used to apply a matrix over the image and simplify sections to highlight the most detailed attributes, these are set in descending in multiple layers in order to enhance the image details effectively. Dropout has been placed between each segment of layers to remove any poorly performing models throughout execution.

Lastly a Flatten is used to change the data shape for the dense layers to apply multiplication matrices to the model to alter the output layer. The final Dense layer output is set to 8 due to the classification amount. The final accuracy achieved was 66.632% which was a major improvement!

Model

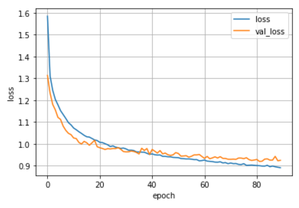
```
model = keras.Sequential(
    [
        Resizing(32,32),

        tf.keras.layers.RandomFlip(),

        keras.Input(shape=input_shape),

        Conv2D(128, kernel_size=(3, 3), padding = "same", activation="relu"),
        Conv2D(128, kernel_size=(3, 3), padding = "same", activation="relu"),
        MaxPooling2D(pool_size=(3, 3)),
        Dropout(0.3),

        Conv2D(128, kernel_size=(3, 3), padding = "same", activation="relu"),
        Conv2D(128, kernel_size=(3, 3), padding = "same", activation="relu"),
        MaxPooling2D(pool_size=(3, 3)),
        Dropout(0.3),

        Conv2D(64, kernel_size=(3, 3), padding = "same", activation="relu"),
        Conv2D(64, kernel_size=(3, 3), padding = "same", activation="relu"),
        MaxPooling2D(pool_size=(3, 3)),
        Dropout(0.3),

        Flatten(),

        Dense(128, activation="relu"),
        Dense(128, activation="relu"),
        Dropout(0.5),

        Dense(64, activation="relu"),
        Dense(32, activation="relu"),
        Dropout(0.5),

        Dense(8, activation="softmax"),
    ]
)
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
history = model.fit(train, validation_data = validation, batch_size=batch_size, epochs=30, callbacks=[checkpoint,callback])
```

Results

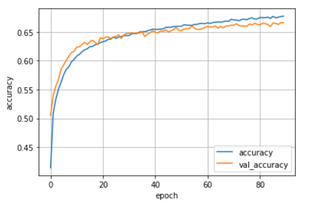
CNN Model	Accuracy
Final	66.63%

Parameters

Input Shape	28
Batch Size	500
Data Split	80/20



For this model, the loss value consistently decreased over each epoch however the val_loss (model tested on 20% random split) seemed to begin to plateau around the 40th epoch only decreasing slightly when reaching the final epoch.



A similar trend is seen on the accuracy data with it consistently increasing, but the val_accuracy correlated closely until the 25th epoch where it began to plateau but still slowly increasing till the final epoch.

Summary

In summary although I found good research leading me towards creating a model using transfer learning the dataset was not viable for creating a accurate model with my highest accuracy at 48.54% mainly as a result of the image size being smaller than most pre-trained models input size. This result lead me to use my second classification method of using a Convolution Neural Network to use the large amount of data to train a model from scratch which resulted in my highest accuracy of 66.632%.

However it is possible I could of found a pre-trained model that could use this small image size and I augmented the data to get around the unbalanced data, using transfer learning has a large amount of parameters which resulted in long epoch times and ultimately my time was better spent perfecting a CNN model as the dataset was more viable for that implementation due to the quantity and compressed image size.

Glossary Of Terms

Convolutional Layer — This layer is used to extract the detailed portions of the image by filtering over the image input and extracting the highest level of detail in the image (upGrad blog, 2020) .

Pooling Layer — This layer normally comes after a convolution layer to reduce the size of the convolved feature map to increase performance and optimize computation costs, there are a variety of types of pooling that have different effects on the feature map (upGrad blog, 2020) .

Dropout — This layer is used when a model is overfitting by reducing the total number of models to increase accuracy and performance (upGrad blog, 2020) .

Dense Layer — This layer receives all the input from the neurons in the layers before and feeds is condensed into this single neuron allowing for a deeper connected layer for learning (Convolutional Neural Network (CNN, 2019) .

Activation Functions — These are used to compute and learn the relationship between variables in the model, there are multiple type of functions all with their own usage from output quantity to binary classification (upGrad blog, 2020) .

Overfitting — This is when a model fits with the training data however this is bad as it can be used to extrapolate data from the model in reference to the initial data (IBM Cloud Education, 2021) .

References

1. Ali, K., Shaikh, Z.A., Khan, A.A. and Laghari, A.A. (2022). Multiclass skin cancer classification using EfficientNets – a first step towards preventing skin cancer. *Neuroscience Informatics, 2(4), p.100034. doi:10.1016/j.neuri.2021.100034.*

2. Stewart, M. (2019). *Simple Introduction to Convolutional Neural Networks. [online] Medium. Available at: https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac*

3. Team, K. (n.d.). *Keras documentation: EfficientNet B0 to B7. [online] keras.io. Available at: https://keras.io/api/applications/efficientnet/.*

4. Convolutional Neural Network (CNN (2019). *Convolutional Neural Network (CNN) | TensorFlow Core.* [online] TensorFlow. Available at: https://www.tensorflow.org/tutorials/images/cnn.

5. upGrad blog. (2020). *Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network*. [online] Available at: https://www.upgrad.com/blog/basic-cnn-architecture/#:~:text=There%20are%20three%20types%20of.

6. Brownlee, J. (2021). *How to Choose an Activation Function for Deep Learning*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/.

7. Shamila Ebenezer, A., Deepa Kanmani, S., Sivakumar, M. and Jeba Priya, S. (2022). Effect of image transformation on EfficientNet model for COVID-19 CT image classify cation. *Materials Today: Proceedings*, [online] 51, pp.2512–2519. doi:10.1016/j.matpr.2021.12.121.

8. TensorFlow. (n.d.). *tf.keras.layers.RandomFlip | TensorFlow Core v2.8.0*. [online] Available at: https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomFlip [Accessed 11 May 2022].

9. IBM Cloud Education (2021). *What is Overfitting?* [online] www.ibm.com. Available at: https://www.ibm.com/cloud/learn/overfitting.