COMP 309 — Machine Learning Tools and Techniques Project: Image Classification

Kamonchanok Suban Na Ayudtaya

1. Introduction:

1.1 Purpose:

Machine learning (ML) is known to be a powerful tool when classifying images. Important abilities for a ML classifier is the classification of a targeted image despite the presence of other objects and multi-purpose classifications capable of classifying different classes of object [1]. The following report aims to investigate methods of validation, loss functions, activation functions, optimisation techniques and hyper-parameters for classifying images containing cherries, strawberries, or tomatoes. A selection of loss functions, activation functions, optimisation techniques and hyper-parameters will be considered 'good' if the model produced is able to achieve and accuracy significantly greater than a simple multi-layer perceptron (MLP) baseline model. The model produced must also be able to be trained in a reasonable time (less than 1 hour).

1.2 Problem:

Keras will be used to create a convolution neural network (CNN) which will allow us to investigate the methods and functions mentioned above.
This CNN will be trained with and tested against a single-label dataset of 6000 images downloaded from Flickr. Of those 4500 (even split between cherries, strawberries, and tomatoes) we will have access to. The CNN must then produce a model that can achieve high classification accuracy on a hidden data set of 1500 images. Widely used loss functions, activation functions, and optimisation techniques should be appropriate for any single label multi-class classification problem on a small dataset without transfer learning, given the appropriate turning of hyper parameters.

1.3 Scope:

This report investigates the best methods and functions to implement a Keras CNN - considering time, as well as self-building the model rather than using transfer learning to better allow for analysis of the findings.

2. Background:

2.1 Theory:

An important aspect of CNN for classification of images is feature extraction. In CNN models feature extraction is computed by convolution layers. These layers consist of filters, logistic function, and pooling operators to reduce the computational cost of training [2].

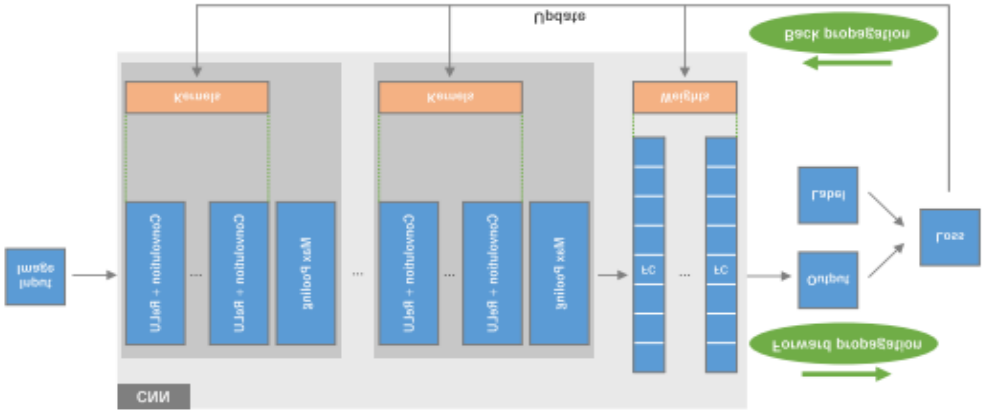

Fig 2.1.1: An overview of a convolutional neural network (CNN) architecture and the training process.
Source: Adapted from [3].

The feature extraction portion of the CNN is depicted in Figure 2.1.1 within the darker grey areas, containing Convolution + ReLU and Max Pooling. It is here that activation functions (such as ReLU) is applied in the model. Activation functions are used after each convolution operation to introduce non-linearity. Without activation functions all the convolution operations would collapse into a single operation. The loss function determines the difference between the expected output and the actual output and is commonly used for back propagation to optimise parameters such as weights (Figure 2.1.1).

2.2 Research:

"Systematic evaluation of CNN advances on the ImageNet" compares different activations functions as well as the results of having no non-linearity on ImageNet to show that having no non-linearity function results in significantly [4]. "On Loss Functions for Deep Neural Networks in Classification" analyses the use of common loss functions in terms of theoretical effectiveness and training time [5]. There is also research being done into new loss functions to improve solutions to problems where more common loss functions are less effective [6, 7]. There has also been recent research into optimal optimisation methods for image classification problems indicating that Adam can be an optimal choice for single label multiclass classification problems [8, 9].

3. Methodology:

3.1 Apparatus

3.1.1 Hardware:

Processor:	1.6 GHz Dual-Core Intel Core i5
RAM:	8 GB 1600 MHz DDR3

3.1.2 Software:

OS:	macOS BigSur (11.6)
Language:	Python v3.7
Libraries:	TensorFlow
	Keras

3.2 Exploratory Data Analysis:

3.2.1 Manual exploration

Before attempting to create and optimise a model, it is important to understand the data. The brief outlines that there are 4500 single class images (1500 of each class) taken from Flickr. All the images had been resized to 300x300 and labelled based upon the tags the image had on Flickr. Due to the source of the images it was highly likely that some images would be miss-classified. To ensure that the dataset was fit for use, I manually scrolled through each image category.

In my exploration I had found that there were images that were not fruits or true fruits (cartoon). Additionally, not all images were 300x300 as outlined by the brief and some images were black and white.

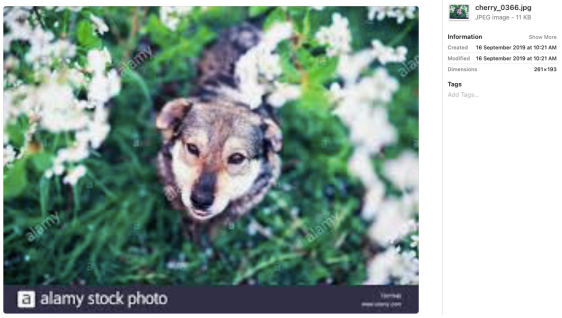

Figure 3.2.1.1: Image taken from Flickr, labelled as Cherry with the dimension 261x193.

To remedy this I manually scrolled through the images and removed images which did not contain fruit as seen in figure 3.2.1 and images which only contain the image labels (e.g the word 'strawberry'). However I did maintain images which were not true fruits (cartoons), black and white images and images with incorrect dimensions. This is because cartoons still depict fruit (model that is limited is not particularly useful assuming test images contain cartoon images), incorrect dimensions can be corrected through preprocessing and colour is unlikely to be a significant feature considering all category of fruits are red.

3.2.2 Morphological transformation

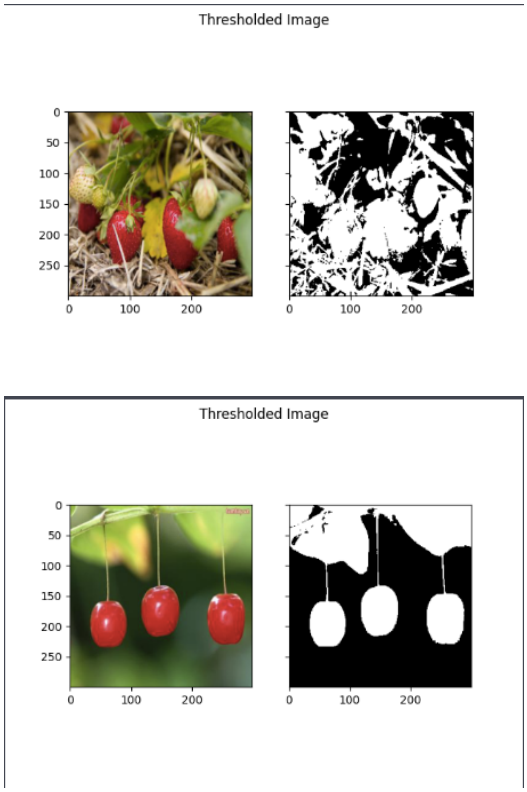

Figure 3.2.2.1: Images taken from Threshold processing result.

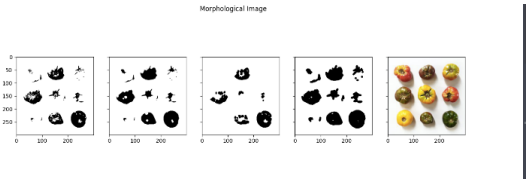

Figure 3.2.2.2: Images taken from Morphological processing result.

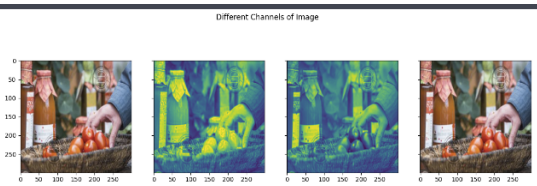

Figure 3.2.2.3: Images taken from channels processing result.

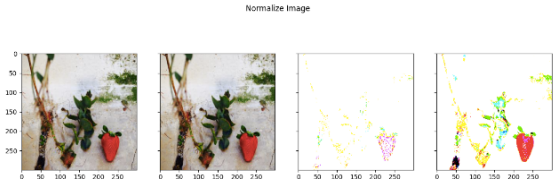

Figure 3.2.2.4: Images taken from normalise processing result.

To try to distinguish the fruit object from the background I have tried multiple morphological transformation methods such as thresholding (binary image: we take all the pixels whose intensities are above a certain threshold, and convert them to ones the remaining to zero), morphology (Erosion, Dilation, Opening & Closing) and Normalisation. It appears that some forms of morphological transformation may help us distinguish the fruit object in some instances however many of the images do not have a clear foreground and background or the fruit is in the background which makes this not particularly useful for this data set.

3.3 Pre-processing:

3.3.1 Remove Noise

The first step we took as discussed above was to remove noise (images which were not fruit) as discussed above. The instances which contribute a low amount of information (provides little value for distinction between fruit) create unnecessary noise (confusion). Removing non-informative features can reduce noise, and can increase the contrast between image categories.

3.3.2 Rescale (normalisation)

The function of 're-scale' is to multiply each pixel value of the image by this scaling factor. In some models, entering the pixel value of the original image may fall into the activation function of 'death zone'. This is observed in relu, where negative value are set to 0, resulting in loss of information. Therefore we have set the scaling factor to 1/255, which scales the pixel value between 0 and 1 facilitating the convergence of the model and helps us avoids loss of neurons (information).

3.3.3 Augmentation

Pooling increases the invariance. If you train a model only using a picture of a strawberry in the top left corner - it will have difficulty dealing with variation for example identifying a strawberry in the right corner augmentation of data consist of flipping, rotation and cropping the model learns all these variations. This significantly boosts the accuracy of the model. So, even if the strawberry is in any corner of the image, the model will be able to recognise it with high accuracy.

3.4 Procedure:

3.4.1 Data split

I split my dataset into training and validation sets in 8:2. The validation set is to obtain useful result where we are able to compare how our model was performing. The validation set provide an unbiased estimation which fit the training set, in this way we can check whether it is overfitting or not.

3.4.2 Loss function

The loss function I used in my model is 'categorical_crossentropy', this is because the purpose of my model is to classify the images into 3 categories (categorical format). Thus it has greatly improved the performance of models with sigmoid and softmax outputs, as it had previously suffered from saturation and slow learning when using the mean squared error loss. After using the 'categorical_crossentropy' loss, the target for each sample should be a 3 dimensional vector that is all-zeros except for a 1 at the index corresponding to the class of the sample. A model that predicts perfect probabilities has a cross entropy of 0.

Baseline (MLP)

To create a model to compare the results of this investigation with, we must develop a baseline MPL model. This model was generated using the minimum

required layers with the default activation, and loss functions.

```
def train_MLP(valid, train):
    model = Sequential()
    model.add(Input(shape=(300, 300, 3)))
    model.add(Flatten())
    model.add(Dense(350, activation='relu'))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    history = model.fit(train, epochs=10, verbose=1, validation_data=valid)
```

Figure A: Snapshot of code used to generate MLP model.

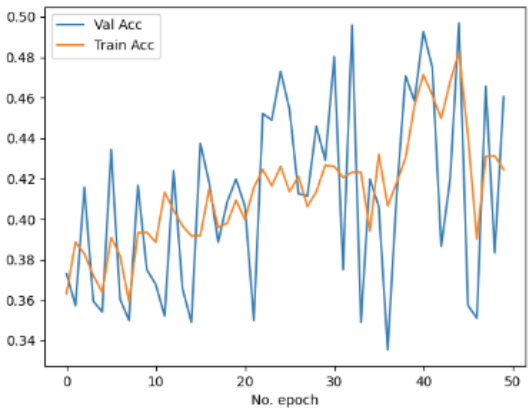

Figure B: Snapshot of accuracy of MLP model.

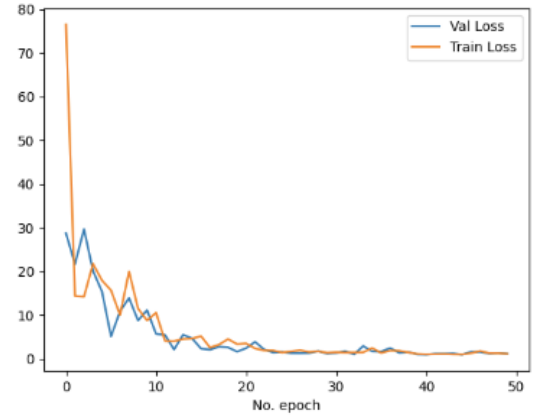

Figure C: Snapshot of loss function of MLP model.

From figure B and C we can observe that there is no clear trend which suggest that the model is randomly allocating images. The best accuracy we obtained was 0.4969 which means the model was not particularly effective at correctly identifying fruits associated to an image - however we were obtaining better results as the number of epoch increase so if we were to increase the number of epoch we may see improved performance in the MLP model.

Basic CNN Model:

```
def basic_CNN(valid, train):
    model = Sequential()
    model.add(Input(shape=(300, 300, 3)))
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
    history = model.fit(train, epochs=25, verbose=1, validation_data=valid)
    plot_result(history)
```

Figure D: Snapshot of code used to generate Basic CNN model

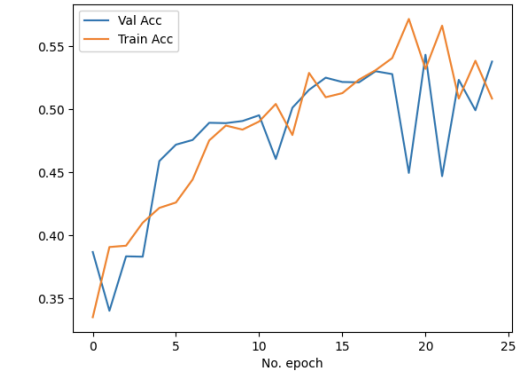

Figure E: Snapshot of accuracy of Basic CNN model.

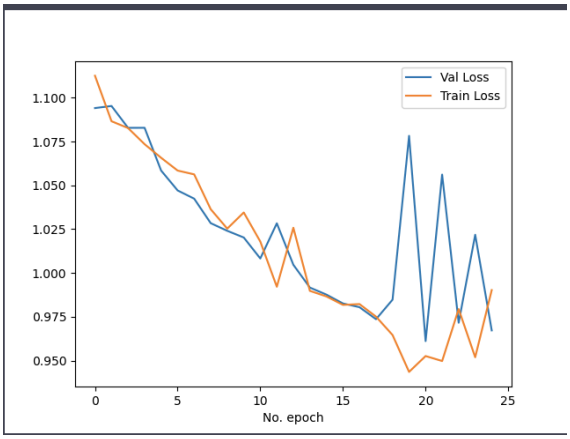

Figure F: Snapshot of loss of Basic CNN model.

Our basic CNN model performed more accurately than our MLP with a clear upward trend with accuracy as the number of epoch increases (downward trend of loss as the number of epoch increases) as observed in figure E and F. The best accuracy obtained was 0.6292 suggesting CNN is a stronger model than MLP.

>Loss graph depicting result - above model is from model with lower epoch.

Optimised model

The CNN model found to be the most effective for the given task had four convolution layers and two dense layers. The model implemented the optimiser Adam, and the loss function sparse categorical cross entropy (Figure 3.4.5.1). The compile time for is model was 2.5 hours and yielded an accuracy of

```
# Train optimised CNN model
def Optimised_CNN(valid, train):
    model = Sequential()
    # 1st Convolution layer
    model.add(Input(shape=(300, 300, 3)))
    model.add(Conv2D(16, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(4, 4), strides=2))
    # 2nd Convolution layer
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(4, 4), strides=2))
    # 3rd Convolution layer
    model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
    # 15% dropout
    model.add(Dropout(0.15))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=3))
    model.add(Flatten())
    # Classification layer
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.15))
    # Second dense layer with softmax activation function to return probabilities of classification
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.001), metrics=['accuracy'])
    history = model.fit(train, epochs=50, verbose=1, validation_data=valid)
    plot_result(history)
```

Figure G: Snapshot of code used to generate CNN model

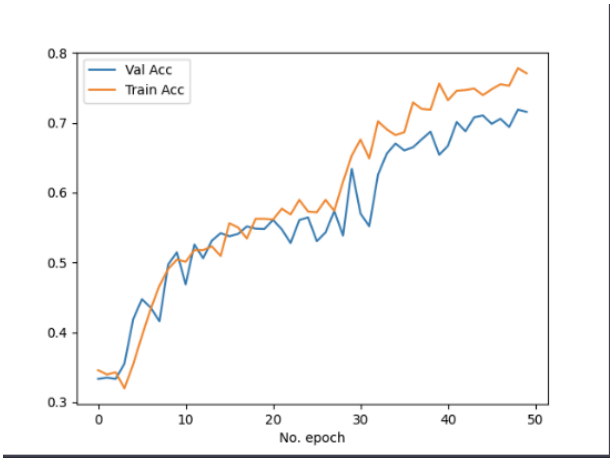

Figure H: Snapshot of accuracy of CNN model.

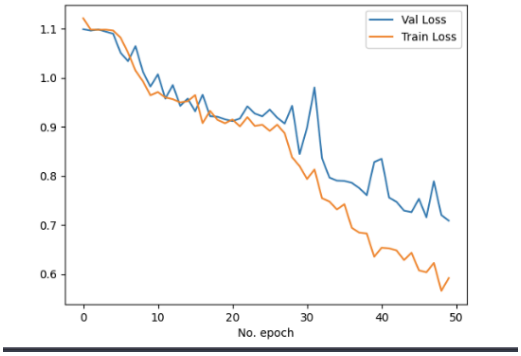

Figure I: Snapshot of loss function of CNN model.

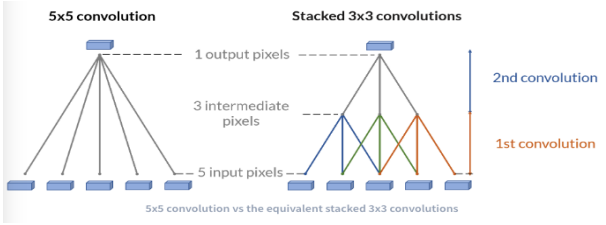

Figure J: Difference between 5x5 kernel and stacked 3x3 kernels with 5x5 input pixels.
Source: Adapted from [10].

Convolution layers

When selecting the number of convolution layers to implement I found that four convolution layers with output features, 16, 32, 64, and 128 respectively to result in good accuracy while keeping the number of parameters in the model low enough that the training time is still reasonable. For each of these layers I used a kernel size of 3 by 3 which is a common choice for image classification according to research conducted in the past. Recent research also indicates that having smaller kernel size with more convolution layers is both better in terms of accuracy as well as number of parameters (Figure 3.4.5.2) [10].

After each convolution layer I have Max Pooling with 4x4 kernels and a stride of 2 which seemed to result in good accuracy and reduced the number of parameters enough to have a reasonable training time compared to Max Pooling with a stride of 1.

3.4.3 Optimiser function

Optimiser can help to minimise the loss value. For the optimisation function used by the model to update parameters during training, the default selection was Adam. This is a stochastic gradient-based method of optimisation that requires only first-order gradient and minimal memory [8]. Once adding convolution layers to my baseline MLP Adam seemed to work as an effective optimiser for the given problem. This choice is backed up by recent research into optimal optimisation methods for image classification as discussed in section 2.1 [9].

3.4.4 Regularisation strategy

Large neural nets trained on relatively small datasets can overfit the training data. This creates an environment where the model is learning the statistical noise in the training data, which results in poor performance when the model is evaluated on new data, e.g. a test dataset. Dropout is a simple way to prevent neural networks from overfitting. Because the outputs of a layer under dropout are randomly subsampled, it

has the effect of reducing the capacity or thinning the network during training. This makes the network more sparse, forcing the neurons to learn more robust features that are useful in combination with many different random subsets of the other neuron and forces the neurons to make the predictions of the missing information.
To do this I have added a 15% dropout after the last two layers, setting 15% of the weights to 0 seemed to reduce over fitting and improve accuracy on the testing set.

3.4.5 Activator function

For each convolution layer an activation function is required as previously discussed in section 2.1. Traditionally activation functions such as sigmoid and hyperbolic tangent (tanh) are used, however more commonly the activation function ReLU is used due to computational efficiency and alleviation of overfitting by reducing the inter-dependency between weights [11]. Using ReLU worked well for this problem despite the disadvantage of a saturated negative region resulting in some weights not being updated during back propagation. The solution to this is Leaky ReLU however, for this problem the use of Leaky ReLU resulted in no improvement for classification.

The activation function selected for the final layer is a softmax function which normalises output values from the last layer to target class probabilities which can help indicate how certain the model is about an image. The use of a softmax layer means that the neural network will now be able to determine the probability that the cherry is in the image, as well as the probability that strawberry and tomato are included as well. And this is also why the value of last dense layer is 3.

3.4.6 hyper-parameter settings

Hyper-parameters are parameters in the model such as the number of epochs, batch size, the learning rate of the optimiser, activation function etc. It is important to tune these parameters to optimise the training process which can help to reduce the time taken to train a model.

I found that the number of epochs being used was important in optimising the relationship between the training set a validation set and this parameter required a lot of tuning. The optimal number of epoch is determined by validation and training loss. As long as it keeps dropping training should continue. For instance, if the validation error starts increasing that might be an indication of overfitting. You should set the number of epochs as high as possible and terminate training based on the error rates. After experimenting it was found that 50 epoch was a good place to stop before overfitting occurred.

Batch size is how many samples we chose to include in one learning step. I found the less it is the faster program can run but blind reduction will cause difficultly in loss function convergence. I chose 4 as my batch size which I've found to be fast and efficient.

In regards to the optimiser, I tried changing parameters such as the learning rate, decay, betas and epsilon and I also tried different optimiser. However, I found when I used the Adam optimiser, the default settings is the optimum settings for my model.

3.4.7 Enrich dataset additional images obtained

I added approximately 350 images replacing those I had removed in the preprocessing and increasing the total dataset size to 4800. These images were the first to appear in a google image search and were manually added to the data set.
This increased the quality of our dataset and allowed for the data to be trained on a greater amount of instances.

3.4.8 Existing models

I attempted to create the VGG16 model however it is extremely time consuming. Although research suggest that it is may be a more accurate model, it is unreasonable in application with a construction time of 5 hours and a compiler too large for my device and a larger to process.

```
# Train existing model
def Existing_model(valid, train):
    K.clear_session()
    # InceptionV3 model and use the weights from imagenet
    conv_base = VGG16(weights='imagenet', input_tensor=Input(shape=(300, 300, 3)),
                      include_top=False)

    VGG = conv_base.output
    pool = GlobalAveragePooling2D()(VGG)
    dense_1 = layers.Dense(512, activation='relu')(pool)
    output = layers.Dense(3, activation='softmax')(dense_1)

    # Define/Create the model for training
    model_VGG = models.Model(inputs=conv_base.input, outputs=output)
    # Compile the model with categorical crossentropy for the loss function and SGD for the optimizer with the learning
    model_VGG.compile(loss='categorical_crossentropy',
                      optimizer=SGD(lr=1e-4, momentum=0.9),
                      metrics=['accuracy'])

    history = model_VGG.fit(train, epochs=50, verbose=1, validation_data=valid)
    plot_result(history)
```

4. Findings:

4.1 Data:

Model Type	Loss	Accuracy	Time(hr)
MLP	1.0258	0.4969	2.5
Basic CNN	0.8771	0.6292	2.5
Optimised CNN	0.5074	0.8333	2.5

While training of both CNN models took significantly longer than the MLP, neither took more than 3 hours to train on the device specified in section 3.1. This is a reasonable time frame that would allow for several trainings with adjustments to hyper parameters and resulted in significantly better accuracy.

4.2 Interpretation:

As seen is section 4.1 the accuracy of the Optimised CNN (83.33%) is significantly better than the accuracy of the MPL (~49.69%). When we look at the loss for both methods we see a similar trend with a significant reduction of loss in the Optimised CNN (~0.5074) from the loss in the MPL (~1.0258).

From this investigation we can discuss with reasonable certainty that the increased performance of he CNN is predominantly due to the convolution layers which allows for extracting high level features from the images. These features are in the case of this problem more informative as to which class the image should be classified as.

Based upon the results of this study for the multi-class single label images classification task completed the most widely used activation and loss functions, ReLU and Adam worked effectively, combined with sparse categorical cross entropy for optimisation. It is important to note however that for the model to be trained well and within a reasonable time frame selecting appropriate hyper-parameters such as kernel size and strides for Max Pooling is important.

5. Conclusion:

5.1 Assessment:

Based upon the finding in this report it is found that generic multi-class single label problems (image classification into simple categories) can be solved well and efficiently using the widely used functions ReLU and Adam with a basic CNN framework. Being able to solve generic classification problems with small datasets within a reasonable timeframe without relying on transfer learning, while only needing to play with and adjust hyper-parameters would open up the accessibility of image classification and lower understanding required to create useful classification models.

6. Recommendations:

Based upon the findings of this study it would be good to experiment with dataset size - identify how small of a dataset can be used to generate good (greater than ~80%) accuracy models.

Another opportunity would be to investigate whether multi-label classification problems can also be generalised in a way that would require only the adjustment of hyper-parameters of the problem outlined by this report.

A final opportunity would be to further explore exisiting models on a more robust machine to potentially produce a more efficient model.

References:

[1] Y. Lin, F. Lv, Zhu, S., M. Yang, T. Cour, K. Yu, L. Cao and T. Huang, “Large-scale image classification: fast feature extraction and svm training,” *In IEEE CVPR*, pp. 1689-1696, 2011.

[2] A. Romero, C. Gatta and G. Camps-Valls, "Unsupervised Deep Feature Extraction for Remote Sensing Image Classification," in *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 3, pp. 1349-1362, March 2016.

[3] R. Yamashita, M. Nishio, R. K. G. Do, K. Togashi, “Convolution neural networks: an overview and application in radiology”, in *Insights into Imaging*, June 2018.

[4] D. Mishkin, N. Sergievskiy, J. Matas, “Systematic evaluation of CNN advances on the ImageNet”, 2016
URL: https://arxiv.org/pdf/1606.02228.pdf

[5] K. Janocha, W. M. Czarnecki, “On Loss Functions for Deep Neural Networks in Classification”, *Theoretical Foundations of Machine Learning*, Feb 2017.
URL: https://arxiv.org/pdf/1702.05659.pdf

[6] J. Fei, T. Rui, X. Song, Y. Zhou, S. Zhang, “More Discriminative CNN with Inter Loss for Classification”, *in Artificial Intelligence and Robotics*, Jan 2018.

[7] Q. Zhu, P. Zhang, X. Ye, “A New Loss Function for CNN Classifier Based on Pre-defined Evenly-Distributed Class Centroids”, Apr 2019.

[8] D. P. Kingma, J. L. Ba, “ADAM: A METHOD FOR STOCHASTIC OPTIMISATION”, *ICLR 2015*, May 2015.

[9] G. Suresh, V. Gnanaprakash and R. Santhiya, "Performance Analysis of Different CNN Architecture with Different Optimisers for Plant Disease Classification," *2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS)*, Coimbatore, India, 2019, pp. 916-921.

[10] Arnault, “About Convolutional Layer and Convolution Kernel”, Oct. 31, 2018. Accessed on: Oct. 29, 2019. Available: https://www.sicara.ai/blog/2019-10-31-convolutional-layer-convolution-kernel

[11] U. Udofia, “Basic Overview of Convolutional Neural Network (CNN)”, Feb. 31, 2018. Accessed on: Oct 27, 2019. Available: https://medium.com/dataseries/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17

[12] L. Perez, J. Wang, “The Effectiveness of Data Augmentation in Image Classification using Deep Learning”, Dec. 31, 2017. Accessed on: Oct. 29, 2019. Available: https://arxiv.org/abs/1712.04621