

AI and ML

(6CS012)

**Marvel Superheroes Image Classification with**

**Convolutional Neural Network**

Student ID: 2329200

Student Name: Aashan Bajgain

Group: L6CG1

Tutor: Mr. Siman Giri

Module Leader: Mr. Siman Giri

Submitted on: 5/14/2025

**Abstract**

The project was designed to develop an image classification system that could classify images of the eight different Marvel superheroes from mere visual input using CNN. The first objective of the project was to create a good classification model. The dataset comprised approximately 2584 images, on which we did some real-time augmentation by means of the ImageDataGenerator class for better generalization of the model. Pre-processing of the images was done through resizing as well as normalizing their pixel values before feeding them to the model. Three CNNs were used in this project: a simple model with fewer layers, a deeper model with more layers and regularization, and transfer learning using a MobileNetV2 pre-trained network. For training, infecting some SGD and Adam over-wise, we compared their performances. ReLU was used in the hidden layer while SoftMax was used in the output layer for the multi-class classification in the first two models that we considered in this project.

The initial model consisted of three convolutional layers with four dense layers and had no dropout or batch normalization involved. For a deeper learning of better generalization and with less chance of overfitting, the deeper network had eight convolutional layers with kernel regularization, dropout, and batch-normalization-based training. The transfer learning setting utilized MobileNetV2, while it was pre-trained using the ImageNet. Initially, all layers of MobileNetV2 were frozen to exploit the pre-trained weights; now, it is the fine-tuning approach that is chosen by unfreezing some number of layers. That transfer learning method really outperformed all other models. Even though the deeper CNN did perform a little better, the baseline and main models were constrained by the dataset on hand. Adam uses may have resulted in better accuracy and faster convergence than SGD, which can be lower with the fine-tuning strategy with MobileNetV2. Had it been up to me, I would state that due to the low quality of images involved, with fewer chances all these models stand at a testing accuracy of above 60%.

**Table of Contents**

[1. Introduction 1](#_Toc198122161)

[2. Dataset 1](#_Toc198122162)

[2.1 Dataset Source 1](#_Toc198122163)

[2.2 Dataset Details 2](#_Toc198122164)

[2.2.1 Dataset Folder Structure 2](#_Toc198122165)

[2.2.2 Dataset Number of Classes 2](#_Toc198122166)

[2.2.3 Number of Images and Image Resolution in Dataset 3](#_Toc198122167)

[2.3 Preprocessing of Data 4](#_Toc198122168)

[2.4 Dataset Observation 4](#_Toc198122169)

[3. Methodology 5](#_Toc198122170)

[3.1 Data Analysis and Preprocessing 5](#_Toc198122171)

[3.2 Model Building 8](#_Toc198122172)

[3.2.1 Baseline Model 8](#_Toc198122173)

[3.2.2 Deeper Architecture Model 9](#_Toc198122174)

[3.2.3 Fine Tuning a Pre-Trained Model 11](#_Toc198122175)

[4. Results and Observation 12](#_Toc198122176)

[4.1 Baseline Vs Deeper Architecture Model 12](#_Toc198122177)

[4.2 Computational Efficiency 14](#_Toc198122178)

[4.3 Model Training with Multiple Optimizers 15](#_Toc198122179)

[4.4 Challenges in Training 16](#_Toc198122180)

[5. Fine-Tuning a Pre-Trained Model 17](#_Toc198122181)

[6. Conclusion and Future Improvements 19](#_Toc198122182)

[7. Code GitHub Link 22](#_Toc198122183)

[8. References 22](#_Toc198122184)

**Table of Figures**

Figure 1: Dataset Folder Structure .................................................................................. 2

Figure 2: Confusing Images - 1 ....................................................................................... 5

Figure 3: Confusing Images - 2 ....................................................................................... 5

Figure 4: Data Class Distribution ..................................................................................... 6

Figure 5: After Resizing and Augmentation ..................................................................... 7

Figure 6: Extra Testing Images + Predictions................................................................ 20

Figure 7: Confusing Patterns in Thanos's Images - 1 .................................................... 21

Figure 8: Confusing Patterns in Thanos's Images – 2 ................................................... 21

# Introduction

We are in the midst of a changing time in AI and Machine Learning, and computer vision may turn out to be the most promising area. Among many other fields that make use of computer vision, image classification has been applied in autonomous vehicles, security, healthcare, e-commerce, and so on. Large-scale success in image classification rests on deep learning and CNNs (:). CNNs can learn how to identify the features themselves in the input data (i.e., the image) from which to recognize patterns in visual information without human intervention.

In applications such as facial recognition, object detection, marine, and autonomous vehicles, CNNs have been proven to work excellently. Self-driving cars, for instance, would need to classify the type of vehicle it has detected (Sarikan et al., 2017). We are in the process of developing and experimenting with various CNN models for the classification of images of eight Marvel superheroes using training data from a dataset from Kaggle. Starting with a simple baseline CNN, we will go on to report the performance of the built models. Later, we will build and train a more complicated CNN model.

Similarly, a transfer learning approach will be considered for the MobileNetV2 model where the features extracted by the network that was pre-trained on ImageNet will be exploited, with the hope of conducting better classification with fewer training resources-little training. The evaluation metrics are accuracy for training and validation, and loss for training and validation, plus a classification report and confusion matrix for each model. Each model will ultimately be compared against one another with all these metrics.

# Dataset

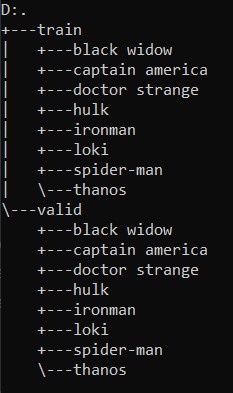
## Dataset Source

* This project will make use of a dataset known as "Marvel Heroes". The dataset is from Kaggle, which is one of the most established data science and machine learning platform to find and share datasets. This dataset was created and published by:
* Name: Ethan
* Username: hchen13

## Dataset Details

### Dataset Folder Structure

The folder structure of the provided dataset is as follows:



This is the folder structure of the dataset. Each folder is labelled with its respective superhero name.

*Figure 1: Dataset Folder Structure*

### Dataset Number of Classes

In the dataset there are 8 classes which are 8 different heroes in the Marvel Cinematic Universe.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Black widow | Captain  America | Doctor  Strange | Hulk | Ironman | Loki | Spiderman | Thanos |

### Number of Images and Image Resolution in Dataset

**Training Data:**

|  |  |
| --- | --- |
| **Class Name** | **Number of Images** |
| loki | 307 |
| hulk | 321 |
| thanos | 323 |
| ironman | 318 |
| doctor strange | 345 |
| spider-man | 326 |
| black widow | 320 |
| Captain america | 324 |
| **Total Training Images:** | **2584** |

**Validation Data:**

|  |  |
| --- | --- |
| **Class Name** | **Number of Images** |
| loki | 54 |
| hulk | 56 |
| thanos | 55 |
| ironman | 56 |
| doctor strange | 61 |
| spider-man | 57 |
| black widow | 55 |
| Captain america | 57 |
| **Total Validation Images:** | **451** |

Smallest Resolution Image: (80, 80)

Largest Resolution Image: (7413, 10617)

Most Common Resolution Image: (1280, 720)

Average Resolution: (897, 734)

Image Type: JPG

## Preprocessing of Data

We generated three drafts for entries of varying sizes and shapes in both areas. The original images in our dataset ranged in size from (80,80) and (7413,10617). We made a modest size reduction to (224,224) in order to create a common baseline. It was necessary to standardize control for pixel size across our dataset and to efficiently process batches of images, while at the same time resizing as a batch of images had better utilization of computing resources, which leads to faster training times, along with using less memory. Resizing the images to this size also positioned us to prepare for well-known pretrained models like MobileNetV2 which specified an input size of (224,224). Due to having a small dataset for deep learning purposes, we took advantage of a few data augmentation techniques to generate synthetic data for size to augment our training dataset. While augmentation parameters will differ from each project, the idea is augment the model's ability to generalize via exploring transformations of the original images. The distribution of images per class was fairly uniform, all classes had relatively about 300 to 350 images, so we also assessed our dataset for class imbalance.

## Dataset Observation

In the initial phase of investigating the dataset, we recognized that the images associated with the different classes were not completely clean. For instance, one image contained multiple superheroes. This could confuse the training and could affect the ability of the model to learn the right distinctions between classes.

A group of people in clothing

AI-generated content may be incorrect.

*Figure 2: Confusing Images - 1*



*Figure 3: Confusing Images - 2*

For instance, in Figure 2, the image has multiple superheroes in it and therefore was categorized into the Black Widow category; but there is also Doctor Strange, Shuri, Ant-Man, and others all mixed into one image.

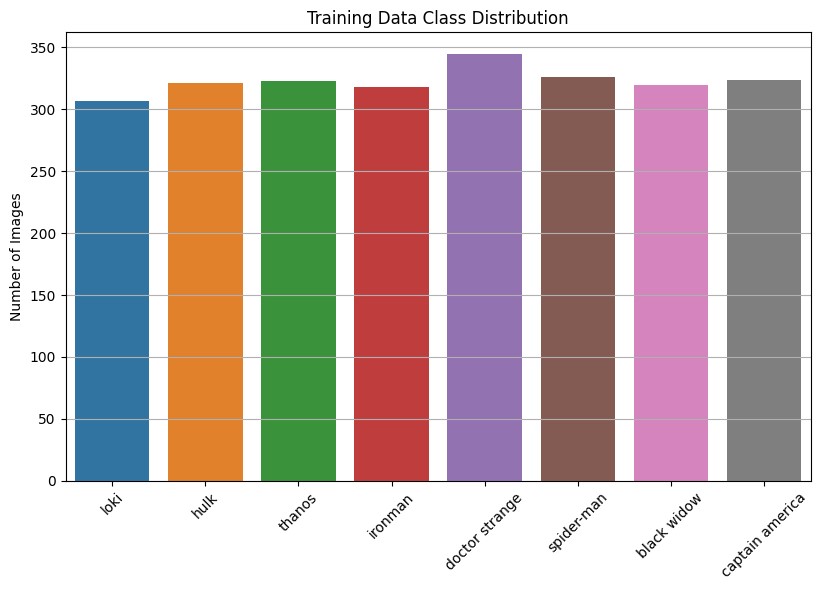
In the same manner, Figure 3 was also defined in the Black Widow category despite containing multiple superheroes, where Captain America is the most prominent. Instances like these can complicate training as having multiple characters present runs the risk of confusing the model and potentially making it more difficult to observe clear categorically specific patterns.

# Methodology

My work in this project consisted of building three models: a baseline model with basic architecture, a deeper model with more complex architecture, and a model with transfer learning.

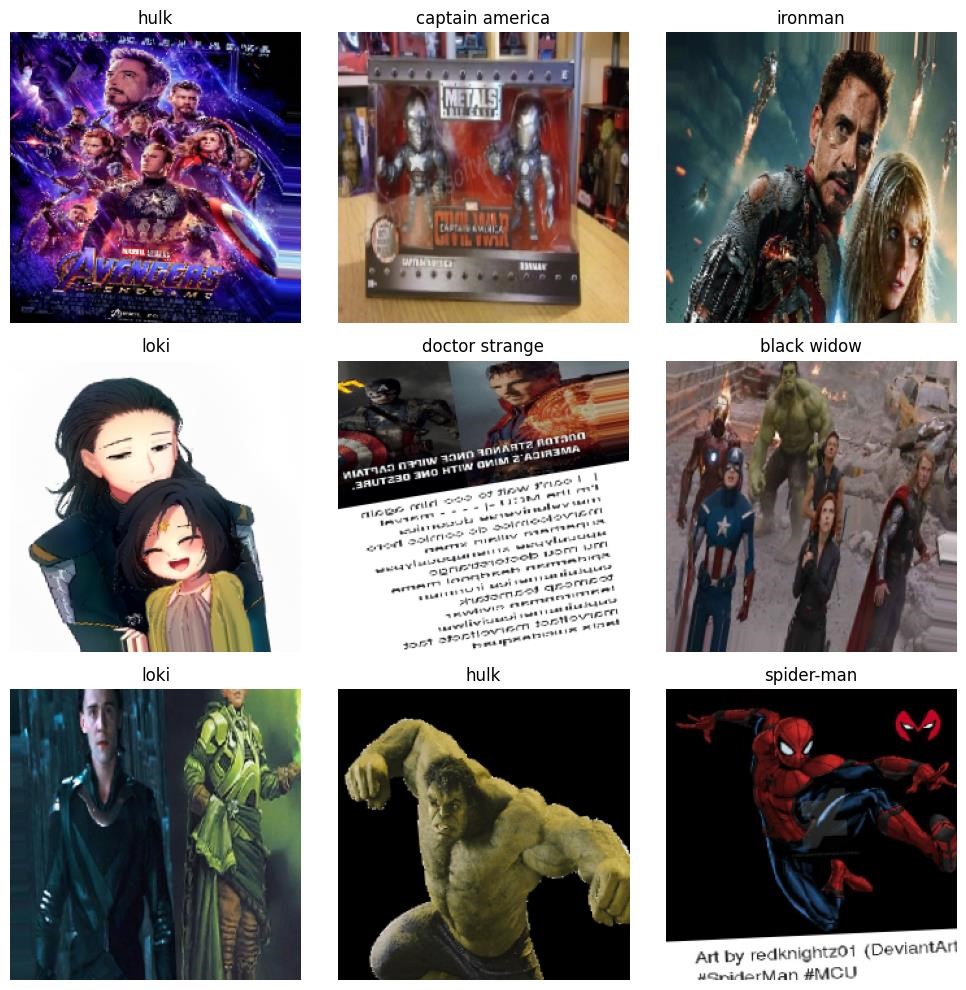
## Data Analysis and Preprocessing

To start the project off, we first worked on inspecting the dataset. That meant inspecting how many images we had, as well as how the dataset was initially configured with regards to training and test splits. We also thought through how we would split the dataset into training, validation, and test splits as part of this initial work.

We came to the conclusion that we would organize the images from the original training folder into an 80% training, 20% validation split. The validation folder would then essentially be the same size as our test set for our end evaluation. Once we completed this analysis, we then moved onto the final data preprocessing steps for the imaging model training cell preparations. Our class distribution analysis indicated that the dataset was quite balanced with approximately 300-350 images per class.

*Figure 4: Data Class Distribution*

To start the project off, we first worked on inspecting the dataset. That meant inspecting how many images we had, as well as how the dataset was initially configured with regards to training and test splits. We also thought through how we would split the dataset into training, validation, and test splits as part of this initial work.  
  
We came to the conclusion that we would organize the images from the original training folder into an 80% training, 20% validation split. The validation folder would then essentially be the same size as our test set for our end evaluation. Once we completed this analysis, we then moved onto the final data preprocessing steps for the imaging model training cell preparations. Our class distribution analysis indicated that the dataset was quite balanced with approximately 300-350 images per class.



*Figure 5: After Resizing and Augmentation*

We utilized the ImageDataGenerator class found in Keras library for the data augmentation aspect of our project. In total, the training data was augmented through a range of transformations that include: rotation, width shifting, zoom, horizontal flips to allow for visual change in the training data and to allow the model to generalize better.

In addition, we also performed a normalization procedure on the image data. The original RGB values ranging from 0,255 were rescaled to a value range from 0 to 1 using the rescale parameter of ImageDataGenerator. We needed to responsively rescale the pixel values to minimize the likelihood that the relatively big range of pixel intensity values would become an issue for training. Normalizing the image inputs will have the effect of more efficient and stable training.

## Model Building

In the time of the project we built 3 models, i.e. baseline model with basic architecture, deeper model (a more complex architecture) and the last model used the transfer learning technique.

### Baseline Model

The baseline model has a total of 11 layers, which consists of three Conv2D layers (with maxpooling layer after each one), a Flatten layer, and a model with an input layer along with four output (dense) layers as fully-connected layers. The model is designed to take images of shape (224, 224, 3), which is RGB images. The main purpose of the model is to classify the image into one of eight distinct Marvel superheroes.

The first layer of the architecture (Conv2D layer) consists of 32 filters (3×3) and serves to extract fundamental visual features (e.g., edges and texture) and actually processes and prepares the image for ensuing layers. A ReLU activation was used as a means of the model providing some non-linearity in the model to learn more complex features. We must also include a MaxPooling2D layer (2×2 window) next that down-samples spatial dimensions of the data for the model to generalize and reduce sensitivity to small distortions in the image.

The second convolution layer has 64 filters, enabling the network to discover more complex patterns. Following this convolution layer is another 2×2 max pooling layer. The third Conv2D layer has 128 filters, used for detecting high level features and structures in the image. It is also followed by another MaxPooling2D layer to down sample the spatial data and preserve the best activation values.

With feature extraction finished, the feature maps are passed to a Flatten layer that transforms all of the multi-dimensional feature maps into a 1D array, allowing the data to pass into classification. The feature maps are then passed through three dense layers with 128, 64, and 32 neurons, each using ReLU activation which the model greatly benefits from learning. The output layer has 8 neurons corresponding to the 8 superhero classes, and uses a SoftMax activation function, which is used here to predict the one class with the highest predicted probability.

**Loss function:** In the baseline model, we used sparse categorical cross entropy as our loss function because the labels are encoded as integers, i.e. in 0 to 7 formats.

**Optimizers:** We used both SGD and Adam optimizers for the baseline model and checked how it would perform in using different optimizers.

**Hyperparameter:** The hyperparameters that we used in baseline model were

|  |  |
| --- | --- |
| **Parameter** | **Values** |
| Learning Rate | 0.01 |
| Batch size | 32 |
| Epochs | 50 |
| Callbacks | Model Checkpoint, Early Stopping |

The hyperparameters chosen to train the baseline model were chosen from the potential hyperparameter choices depending on dataset sizes and model metrics. We chose batch size of 32, as this value provided the best balance of trainability on our dataset sizes while also using available memory. Initially the model was trained for 50 epochs, but because validation accuracy continued to improve, and val\_loss continued to decrease, it appeared that the model had not yet converged. The number of epochs was updated to 100, as this would allow the model as many epochs as possible thereby reducing overall training time as much as possible.

To protect against overfitting we used Early Stopping as an additional precaution. This callback keeps track of the val\_loss during training and allows for the option to stop training if there is no danger in continuing training. For example, we used val\_loss and applied a patience of 10, which meant we'd stop training if val\_loss did not improve for 10 epochs in a row.

### Deeper Architecture Model

After the base model, a much more complex structure was developed by putting many layers over the initial baseline and using additional components such as dropout, batch normalization, and regularization in order to increase performance and generalization.

The model consists of 8 convolutional layers, 4 pooling layers, 11 batch normalization layers, 2 dropout layers on top of a flatten layer, 3 dense layers, as well as the output layer. The architecture as a whole is structured in blocks. The first block has 2 Conv2D layers that each have 32 filters of (3×3) size, accompanied by a Batch Normalization layer and a maxpool layer. The second block has the same structure, but has 64 filters now.

This continues for the following two blocks, which have 128 and 256 filters respectively, to evolve the network to exploit more complex and richer patterns/features. Once the ability to extract features from the convolution is reached a Flatten layer will be used to flatten the map of features from multi-dimensional to a 1-D vector, which will allow the classification process to occur.

At the next layer the network will have three dense layers with 256, 128 and 43 neurons respectively. Each dense layer is followed with a BatchNormalization layer to allow for the learning to stochastic and converge faster. For the first two dense layers, Dropout layers with the corresponding dropout rates of .3 and .2 will be used to reduce the chances of overfitting and improve generalization as well.

**Loss Function:** Similar to the baseline model in the deeper model, we used sparse categorical crossentropy as the loss function because it's target labels were encoded as integers (0 - 7 classes).

**Optimizers:** SGD (Stochastic Gradient Descent) and Adam were the optimizers that employed the deeper model. In SGD momentum is used and set at 0.9 along with the learning rate of 0.001 it helps accelerate the gradients in the right direction and mitigates oscillations. In the Adam optimizer the learning rate is 0.0005.

**Hyperparameters:**

|  |  |
| --- | --- |
| Parameters | Value |
| Learning Rate | SGD – 0.001 and Adam – 0.0005 |
| Momentum (SGD) | 0.9 |
| Epochs | SGD – 100 and Adam - 100 |
| Callbacks | ModelCheckpoint, EarlyStopping,  ReduceLROnPlateau |
| Regularization (L2) | 0.0005 (Conv2D and FC Layers) |
| Dropout Rate | 0.3 and 0.2 |
| Batch Size | 32 |

### Fine Tuning a Pre-Trained Model

The final model is based on a pre-trained model called MobileNetV2 which is based on a pre-training model on ImageNet large dataset. To build the model, we first import the pre-trained model and then remove the top, fully connected layers of the pre-trained model to customize layers based on our use-case, MobileNetV2 is a dataset trained on 1000 classes, while we required a dataset of only 8 classes.

The architecture of a transfer model has a base model (MobileNetV2) and is then followed by a GlobalAveragePooling2D layer to convert the feature map to a flat vector. Then we have two Dropout layers for regularization and better generalization. The next layer is fully connected with 128 units and ReLU activation function. Finally, the last dense output layer has 8 units in a multiclass classification use-case with softmax activation function.

**Loss Function:** In the transfer learning model too, we’ve used the sparse categorical crossentropy loss function because the labels are encoded as integers (0 to 7 classes).

**Optimizers:** In all our transfer model experiments, we have used the Adam optimizer, since we could see that it was the better performing optimizer in both the baseline model and the deeper architecture model.

**Hyperparameters:**

|  |  |  |
| --- | --- | --- |
| Parameters | Value |  |
| Learning Rate (Adam) | Initial Training: 0.0001,  Fine-tuning: 0.00001 |  |
| Epochs | 50 initial, 50 fine tuning |  |
| Callbacks | ModelCheckpoint,  ReduceLROnPlateau | EarlyStopping, |
| Dropout Rate | 0.5 and 0.3 |  |
| Batch Size | 32 |  |

# Results and Observation

## Baseline Vs Deeper Architecture Model

For both the baseline model and the deeper architecture model we first built the architecture for the models and then we trained the model using two optimizers, i.e. SGD and Adam optimizers.

**Baseline SGD Model:** For the baseline SGD model, the epochs (number of times the model ran the entire training data set) were set to 100 for training and watched the training process. The models accuracy started at 0.1254 (12%), with a validation accuracy of 0.1712, then the loss of the model started at 2.0837 and a validation loss of 2.0749. As the epochs continued to increase, the accuracy of the model increased at a constant slow pace and the validation accuracy also increased, with a lower validation loss. The highest validation accuracy obtained by the model was at the 56th epoch, i.e. 0.5233 with the lowest validation loss of 1.3565, but then the model's validation accuracy could not continue to increase, instead the validation loss began to climb after the 56th epoch indicating overfitting. Early stopping callback was use during the model trainining with a patience of 10, so the models training ended after the 66th epoch.

**Deeper SGD Model:** For the baseline SGD model, the epochs (number of times the model ran the entire training data set) were set to 100 for training and watched the training process. The models accuracy started at 0.1254 (12%), with a validation accuracy of 0.1712, then the loss of the model started at 2.0837 and a validation loss of 2.0749. As the epochs continued to increase, the accuracy of the model increased at a constant slow pace and the validation accuracy also increased, with a lower validation loss. The highest validation accuracy obtained by the model was at the 56th epoch, i.e. 0.5233 with the lowest validation loss of 1.3565, but then the model's validation accuracy could not continue to increase, instead the validation loss began to climb after the 56th epoch indicating overfitting. Early stopping callback was use during the model trainining with a patience of 10, so the models training ended after the 66th epoch.

|  |  |  |
| --- | --- | --- |
| **Metrics** | **Baseline SGD Model** | **Deeper SGD Model** |
| Best Accuracy | 0.6489 | 0.5698 |
| Best Validation Accuracy | 0.5195 | 0.5759 |
| Lowest Validation Loss | 1.3565 | 2.0758 |
| Training Time | ~55 minutes | ~46 minutes |
| Epochs | 38/50 | 48/50 |

**Baseline Adam Model:** In the baseline Adam model, we set our number of epochs to 50 assuming Adam optimizer converges faster than the SGD optimizer. The baseline Adam model's test accuracy started from 0.1250 with the validation accuracy of 1829 with validation loss of 1.9807. The validation accuracy increased until epoch 15 and peaked up to 0.5195 with a validation loss of 1.4203 in epoch 15. However, after the 15th epoch the model's training accuracy would increase, but validation accuracy could not go beyond 0.5195 and validation loss couldn't minimize below 1.4203. So in the end, the model's training stopped at 25th epoch, since as described we were using the early stopping callback.

**Deeper Adam Model:** For the more advanced Adam model, we set the number of epochs to 100, which was good in this case since we had an increasing learning rate. The accuracy, validation accuracy, and validation loss for the model at the start of training was 0.1449, 0.1245, and 3.4416. During training and as the number of epochs increased, we saw the accuracy and validation accuracy increase considerably, whilst the validation loss decreased. The validation loss decreased until the 40th epoch, where it reached a value of 1,7104, and a validation accuracy reaching a value of 0.6595. After this point, the validation loss began oscillating but not going below the validation loss of the 40th epoch, and even though we could see the validation accuracy was on a small increasing trajectory. Therefore, the training stopped at the 50th epoch using the early stopping callback function.

|  |  |  |
| --- | --- | --- |
| **Metrics** | **Baseline Adam Model** | **Deeper Adam Model** |
| Best Accuracy | 0.6676 | 0.8529 |
| Best Validation Accuracy | 0.5195 | 0.6790 |
| Lowest Validation Loss | 1.4203 | 1.7104 |
| Training Time | ~55 minutes | ~58 minutes |
| Epochs | 25/50 | 50/50 |

## Computational Efficiency

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Metric** | **Baseline SGD** | **Baseline Adam** | **Deeper SGD** | **Deeper Adam** |
| Training Time | ~55 mins | ~55 mins | ~46 mins | ~58 mins |
| Saved Model Size | 42.6 MB | 127 MB | 59.4 MB | 89 MB |
| Epochs | 66/100 | 25/50 | 48/100 | 50/100 |

As deep leaning/CNN is a resource-hungry machine learning task, we required faster processing time to decrease our training time. We used Google Colab's GPU runtime for faster training of the model.

Prior to using the Google Colab GPU environment, we had attempted training the model in a local and Colab CPU environment, which turned out to be inefficient and lasted a long time to train. This is when we decided to train our model using the T4-GPU runtime in Colab.

Google Colab provided roughly a 4-hour time limit usage in the GPU runtime environment. As it took almost an hour to train each model with different optimizers, it is impossible to launch the training process on a single day because of the time limit to use the GPU, regarding that we had to fit and train four models, change the parameters and refit, not to mention fixing bugs. For these reasons, it took a lot of time to train the models and get the results.

Moreover, our dataset was a relatively complex dataset with complex images and confusing data as mentioned in *2.4 Dataset Observation.*

## Model Training with Multiple Optimizers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Metric** | **Baseline SGD** | **Baseline Adam** | **Deeper SGD** | **Deeper Adam** |
| Training Time | ~55 mins | ~55 mins | ~46 mins | ~58 mins |
| Convergence Speed | 50 epoch | 15 epoch | 37 epoch | 39 epoch |
| Epochs | 66/100 | 25/50 | 48/100 | 50/100 |

We used SGD and Adam optimizers in the model building process on both the baseline and deeper models. In the above table showed that the convergence speed between the baseline SGD and Adam i.e. in the case of SGD optimizer model got convergence at around 55th epoch whereas Adam optimizer model got convergence at around 15th epoch this is significantly less epoch. But in the case of Deeper model we can see the Deeper SGD model has the convergence speed - 37 epoch which means the loss stopped getting reduced after 37th epoch and the Deeper Adam model has the convergence speed of 39 epoch which means the loss stopped getting reduced after 39th epoch. Here we can see the Deeper Adam model took more time to converge than that deeper SGD model.

The most possible explanation of the Deeper Adam model taking more epochs to converge is due to the learning rates of the different optimizers. The Deeper Adam model has a learning rate of 0.0005 whereas the Deeper SGD model has the learning rate of 0.001 which may have resulted in the slower learning of the Adam model therefore making the convergence slower.

## Challenges in Training

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Metric** | **Baseline**  **SGD** | **Baseline**  **Adam** | **Deeper SGD** | **Deeper Adam** | **Total Time** |
| Training Time | ~55 mins | ~55 mins | ~46 mins | ~58 mins | 3hrs 33mins |
| Computational Cost | High | High | Moderate | High |  |

We face different challenges while training the model in our dataset. The first challenge we faced while training was the training time challenge where we had to wait for at least 50 mins for a model to train after switching to the GPU environment of Colab. Before that we had to wait for longer periods for the model to train.

In the initial stage of the model training, when we used the data augmentation, due to the augmentation being too rigorous, the model was unable to capture any patterns while training. Thus, first we tried training the model without augmentation which resulted in model performing better and capturing some patterns, but it was overfitting. After this we got to know that excessive augmentation was the cause of model’s accuracy not increasing and inability of the model to capture patterns. So, we decreased the augmentation of the data to be slightly softer which resulted in better accuracy and better generalization of the model.

Moreover, while training, we faced issues like overfitting in the Adam model which we tackled through deeper architecture and regularization techniques.

When building a deeper architecture, there was a point where the model was unable to learn anything from the dataset because it was too complex of an architecture, along with too much dropout and regularization. Therefore, we had to decrease the drop out rate from 0.5 to 0.3 or 0.2 so that the model could learn the patterns without too much performance degradation.

# Fine-Tuning a Pre-Trained Model

For our project, we chose MobileNetV2 to be the pre-trained model which is a lightweight and efficient model architecture while still being very effective on image classification tasks given limited computational resources.

When creating the transfer learning model we took a two part approach. First, we started out using feature extraction. We froze the base layers of the pre-trained model on the ImageNet dataset so the trained weights were not updated immediately. Then, we added a Global Average Pooling layer for dimensionality reduction above the frozen base model. Next, we added a dense layer of 128 units with a ReLU activation function, and also had two dropout layers for regularization. Lastly, we added a dense layer where the activation function was softmax classifying the 8 classes of images.

We used Adam optimizer for the model as it showed faster convergence and satisfactory results in the deeper architecture model. Also, callback called early stopping was used for minimizing overfitting in the model.

After completing the initial training, now we unfroze the top 20 layers of the pretrained model so that the modal can adapt to the new dataset with its unique features. We also reduced the learning rate to 0.00005 to prevent the model from rigorously updating the weights which can harm the pretrained weights.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Metric** | **Baseline SGD** | **Baseline Adam** | **Deeper SGD** | **Deeper Adam** | **Pretrained**  **Fine Tuned Model** |
| Accuracy | 0.6489 | 0.6676 | 0.5698 | 0.8529 | 0.7037 |
| Validation Accuracy | 0.5195 | 0.5195 | 0.5759 | 0.6790 | 0.8054 |
| Validation Loss | 1.3565 | 1.4203 | 2.0758 | 1.7104 | 0.6530 |
| Training Time | ~55 mins  (GPU) | ~55 mins (GPU) | ~46 mins  (GPU) | ~58 mins  (GPU) | ~49.9 mins  (CPU) |
| Convergence Speed | 50 epoch | 15 epoch | 37 epoch | 39 epoch | 17 Epoch |
| Epochs | 66/100 | 25/50 | 48/100 | 50/100 | 22/50 |

When we compare the pretrained fine-tuned model with the others built from scratch we can see clearly that the transfer learning model has performed much better and generalized. Even in CPU run time it is getting training time the others are getting from GPU. Hence, it also demonstrates that the model is demanding less computational power.

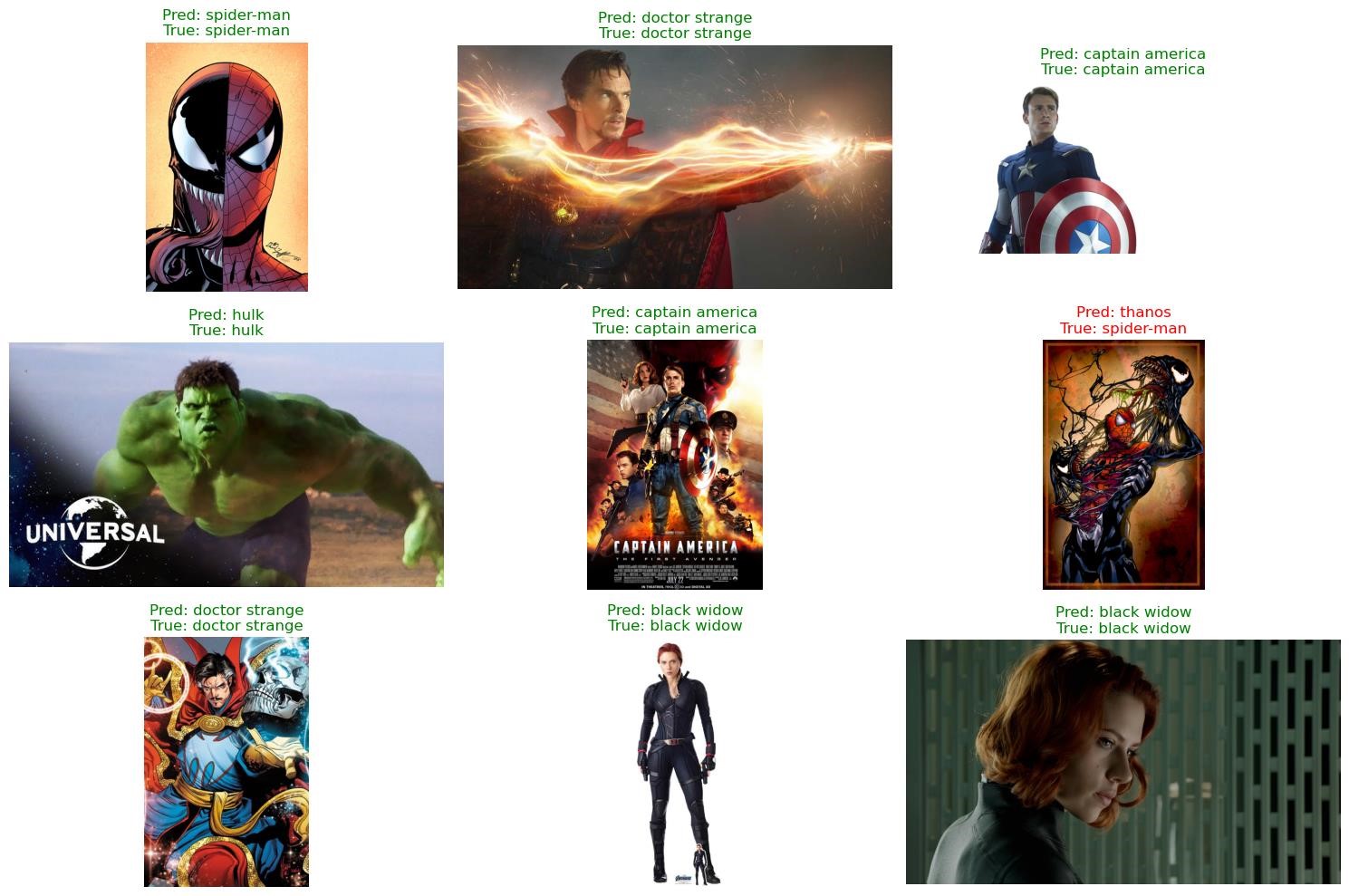
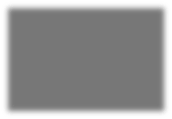
# Conclusion and Future Improvements

Now that we have created 3 separate models with three different optimisers and three different techniques, we have gained a number of insights into the project, dataset and image classification. The first architecture or the baseline model had a good model for a smaller dataset problem but can lead to overfitting, which leads to fine-tuning issues because of the lack of generalisation from the training data. The second architecture was a more complicated architecture, with a larger number of layers in the model, addition dropout layer, kernel regularization, batch normalisation, and so on. This actual model was better generalized, but in terms of model training time, it had a slightly increased training time in comparison to the baseline as it had a larger number of layers.

Finally, the third architecture was a pretrained model that we had fine-tuned. In the third model, we first froze the top 20 layers first to prevent the pretrained weights to update too quickly. This allowed us to model first with pretrained weights while keeping the top 20 layers frozen and later on to unfreeze those layers so the model could learn the features from the diversity of dataset. After we had trained the model, we observed that the pretrained fine-tuned model had learned generalize better and use those learnings to solve the unseen data better.

Since we found that the files in the dataset were misleading and actually contained more than one image of superheroes, in conjunction with our testing dataset, we downloaded some sample images of the 8 superheroes from on the web and tested those images alongside our own model.

We observed that the images with one specific superhero were classified with excellent accuracy, but the images with multiple people were mis-classified . With the new images taken from the internet which contained a total of 22 images of 8 superheroes, accuracy ended up being approximately 86% which is very great accuracy regarding the quality of the dataset.



*Figure 6: Extra Testing Images + Predictions*

Here we can clearly see that the model is predicting the superheroes with decent accuracy, it failed to predict the image that had confusing patterns.



*Figure 7: Confusing Patterns in Thanos's Images - 1*

*Figure 8: Confusing Patterns in Thanos's Images – 2*

In addition, most of the wrong predictions occurred in the "thanos" class because in the dataset images we could see that the image with Thanos contained 6 infinity stone and each one was a different color which might have made it difficult for the model to accurately interpret the patterns.

A possible future improvement to the model would include building a dataset that is much more clean and accurate for training. The fact that images might contain multiple superheroes instead of only containing the exactly labelled superhero really hurt how well the model was able to perform.

If I could obtain a more better new dataset, or a updated dataset with more clean images and better selecing of images could help in training the model with more accuracy.

# Code GitHub Link

GitHub Commit Link:

[Link to GitHub Commit](https://github.com/ShrDar/AI-and-ML/commit/4af9681307ca71fe88bed091b2689e9ed9b46424)

Direct File Link:

[Direct File Link](https://github.com/ShrDar/AI-and-ML/blob/main/Assessments/Assessment%20-%201/2329504_DarshanShrestha_ImageClassificationCNN.ipynb)

# References

Sarikan, S. S., Ozbayoglu, A. M. & Oguzhan, Z., 2017. Automated Vehicle Classification with Image Processing and Computational Intelligence. *Science Direct,* 114(1), pp. 515522.