ANALYSIS OF KERAS DEEP LEARNING ARCHITECTURES FOR GARBAGE CLASSIFICATION

Brandon Quan, Gregory Slowski, Michael Kissinger, Meet Pandya, Zachary Frena

University of Calgary
ENEL645 - Winter 2022
Dr. Roberto Medeiros de Souza
Final Project
(https://github.com/pandyama/ENEL645-SlamDunkers/tree/main/FinalProject)

ABSTRACT

The purpose of this report is to discuss the garbage classification that the team has done using neural network models available in Keras and what kind of accuracy and performance we managed to achieve. We used the garbage dataset that was made available by Dr. Roberto de Souza which has nearly 5000 images. Along with that, we chose 5 models namely VGG16, ResNet50, DenseNet121, EfficientNetB0, and InceptionV3 from the Keras applications library. Each of the 5 members of our group chose a model and worked on training the model in multiple different ways and observing the results to get an understanding of potential ways to do garbage classification. All models performed well with test classification accuracy all falling between 74% and 82%. The best performing model was a DenseNet121 model using transfer learning with Imagenet weights and fine-tuning to achieve a score of 82.15%.

1. INTRODUCTION

Garbage classification is a popular classification problem in the machine learning area. The underlying basis for this problem is that, given a picture of a piece of garbage, an algorithm can correctly classify the image according to the proper disposal requirements for the piece of garbage. The significance and importance of this problem becomes apparent when it is considered through the lens of an environmental problem. In everyday life, when people have difficulty with understanding the different kinds of garbage and the proper bin to dispose of common household items, the incorrect disposal of items can lead to unnecessary items being sent to the landfill when it could be recycled or composted.

A well designed garbage classification algorithm could be implemented in a mobile app for everyday users or it could be used by large companies to accurately segregate their garbage. In this project, our group decided to evaluate this problem by using a variety of models available through Keras. These models include VGG16, ResNet50, DenseNet121, EfficientNetB0 and InceptionV3. These models were chosen based on a number of factors, such as the model's number of parameters and depth. For example, DenseNet121 has 8.1 million parameters with a depth of 242, while VGG16 has 138.4 million parameters but a depth of only 16. By choosing a wide selection of models, the intent of this project is to evaluate the performance of the different models and compare their results against one another.

2. RELATED WORK

Garbage classification is a popular topic in deep learning due to its relevance to everyday life - sorting garbage is a common task with important implications for the environment. Examples of related papers include:

”Common Garbage Classification Using MobileNet” by Radano et al., generates a model and mobile app capable of classifying garbage into six categories - glass, paper, cardboard, plastic, metal, and other [1].

”Carbon emissions under different domestic waste treatment modes induced by garbage classification: Case study in pilot communities in Shanghai, China” by Chen et al., looks at the impact of garbage classification where from a case study of pilot communities in Shanghai, it was estimated that landfill waste could be significantly reduced by 17% [2].

3. METHOD AND APPROACH

In order to train our selected models, each member of the group trained one of the models using different strategies to generate varying results. In general, the first approach was to ensure that we are not using any pre-loaded weights such as ImageNet. There was a curiosity within our group to see what happens when we train our model completely from scratch. These findings will be discussed later in the report within the Results section.

After training without weights, many members of the group chose to evaluate the same model using transfer learning with the weights of the ImageNet dataset included with the intent and ultimate goal of comparing it to the results without the weights.

Slight variations in the initial and fine tuning, number of epochs used, and addition of dropout layers are apparent across the different group members and their strategies in developing their models for this classification problem.

3.1. Code Approach

With regards to the coding approach, the group leveraged the Keras applications library [3] along with the Python programming language. Additionally, a transfer learning tutorial and garbage image dataset provided by Dr. Souza was the basis for much of the project and the strategies used. Cloud computing resources provided by University of Calgary, known as the TALC cluster, were used in the training of the models for this project.

The code was broken down into different sections in a manner consistent with many neural network models trained in industry or by academic individuals. Generally speaking, the dataset was split into a training set, a set for validation, and a final test set for the purposes of evaluating the results of each model. The models would be trained on the training set, validated using the validation set, and then undergo a final evaluation using the test set. The training set consists of 5139 samples (73%), the validation set of 1123 samples (16%), and the test set of 807 samples (11%). The total class distribution is seen in the figure below.

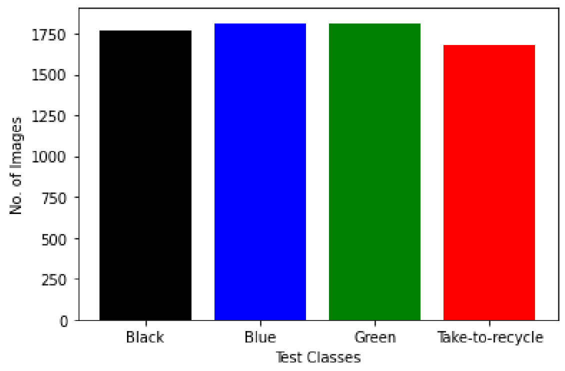

Fig. 1: Class distribution of the garbage classification dataset.

Similarly to the total distribution, all sets are relatively balanced between the four classes and therefore do not require special attention for imbalanced datasets. Overall the working set is 89% of the data, while the test set is 11% of the data which for a relatively large image dataset is a reasonable split, without incurring significant effort to increase the test set size.

3.2. VGG16

VGG16 (sometimes referred to as OxfordNet) is a convolutional neural network architecture named after the Visual Geometry Group from the University of Oxford. Karen Simonyan and Andrew Zisserman developed this architecture through their article "Very Deep Convolutional Networks for Large-Scale Image Recognition" and used it to win the highly prestigious ImageNet Challenge in 2014[4]. The input into the VGG16 network is an RBG image of size (224x224x3), which is followed by 2 convolutional layers and a max pooling layer. This arrangement of 2 convolutional layers plus an additional max pooling layer is repeated once more, after which 3 sets of 3 convolutional layers separated by max pooling layers are added. Finally, 3 fully connected layers are placed at the very end to complete the network structure. In summary, VGG16 consists of 13 convolutional layers and three fully connected layers (summing to a total of 16 total layers, hence the name)[5][6].

The approach of using VGG16 in this project consisted of performing 4 consecutive tests, the results of which were logged and compared against each other to identify the optimal characteristics. Each subsequent test involved the alteration of the VGG16 model parameters, including epoch limits, training weights, and dropout layers. Common to all tests were the learning rates for both initial training (10^{-4}) and fine tuning (10^{-8}). Our first test was to train our VGG16 model from scratch, and not use any pre-determined weights. The model performed very poorly (25.77% testing accuracy), presumably no better than a model that used random chance. Next, we used a transfer learning approach where we used ImageNet weights. The result of this change was drastic– our test accuracy jumped from 25.77% to 70.88%. This transfer learning approach was adopted as the base case for the remaining modeling.

The third modelling attempt used a 50 epoch limit to identify if any additional accuracy gains could be made past the original 10 epoch limit. With a patience of 5 epochs, the VGG16 model with the ImageNet weights reached early stopping after 35 epochs which led to a training accuracy of 90.60% and a validation accuracy of 78.10%. The decision to add epochs was an important one to make because it was clear our model was getting much more accurate the longer it was allowed to iterate through the epochs.

The final model alteration was adding a dropout layer (with a dropout rate of 30%) in order to prevent any potential over-fitting on the training data. The reason to suspect this over-fitting was because we can see a somewhat large discrepancy between the training accuracy (90.60%) and the validation accuracy (78.10%). After implementing this dropout layer, the validation and accuracy scores became closer in value (training accuracy: 86.31% and validation accuracy: 80.70%), and the final testing accuracy score increased slightly when compared to the non-dropout model.

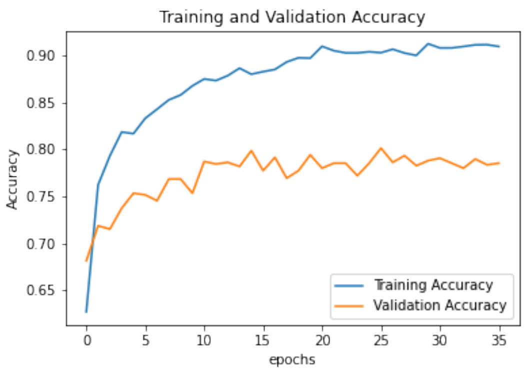

Fig. 2: Train and validation accuracy for VGG16 Transfer Learning after 35 epochs.

This indicates that the introduction of the dropout layer achieved a successful reduction in over-fitting behaviour. In addition, the number of epochs iterated through before early stopping was achieved was also reduced from 35 to 26, reducing overall computational resources.

3.3. ResNet50

ResNet50 (Residual Network 50 layers) is a 50 layer convolutional neural network initially proposed in 2015 by He, Zhang, Ren and Sun [7] [8]. The model is inspired by VGG-19 architecture with additional shortcut connections, which in the ResNet50 architecture specifically create shortcuts that jump 3 layers at a time, rather than 2 in the original architecture which only had 34 layers. [9]. The ResNet50 model is typically used for computer vision like image classification, object localisation, and object detection but can also be applied to other deep learning tasks.

The ResNet50 coding approach from our team was broken down into 3 different steps, typically comparing the result of sequential tests and determining how to proceed after each. All models included both initial training with a learning rate on the order of 10^{-4} and a fine tuning learning rate in the order of 10^{-8}. The first step was comparing a typical transfer learning approach where ImageNet weights were used for the ResNet50 base model, to a standard use of ResNet50 without ImageNet weights. The model performed significantly better with ImageNet weights and therefore the transfer learning approach was adopted for the remaining modeling.

The second step was to use the same transfer learning approach, but with a maximum epoch limit of 100 instead of 10, as early stopping criteria was not reached in either of the first two models using only 10 epochs. Using a 100 epoch limit, early stopping was reached with the transfer learning model after 12 epochs, with a patience of 10 epochs. Realistically this did not add a significant amount of accuracy but continuing with a max of 100 seemed beneficial, and not detrimental to computational resources required due to the early stopping.

The final step was testing dropout which is added to the model after the base layer. Dropout is proven to reduce model over fitting [10], as the ResNet50 transfer learning was showing training accuracy in the range of 95% and validation accuracy in the low 80% range as seen in the epoch accuracy scores in the following figure. This separation indicated some model over fitting where it was determined trying a dropout layer after the ResNet50 base model of 20% and 50%, as recommended by Baldi and Sadowski for maximize the regularization effect [11] could result in increased validation accuracy, and furthermore testing accuracy.

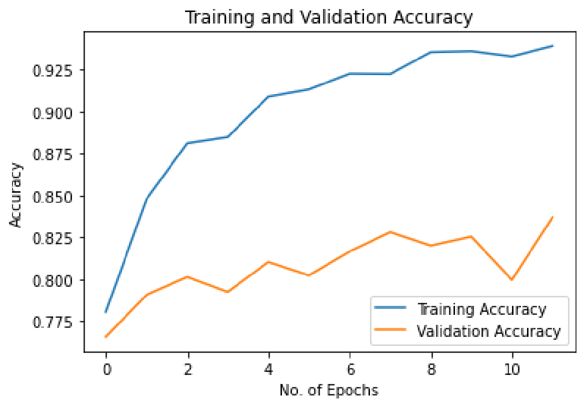

Fig. 3: Train and Validation accuracy for ResNet50 Transfer Learning after 12 epochs

The final model used for ResNet50 was therefore a transfer learning approach using ImageNet weights, with a limit of 100 epochs, and a dropout layer to reduce overfitting from the transfer learning base model, with both initial training and fine tuning of the model.

3.4. DenseNet121

The DenseNet model was proposed in 2018 in the paper titled ”Densely Connected Convolutional Networks” [12]. One of the key distinctions of this model is that for each layer, all of the features of the previous layers and the current layer are used in forthcoming layers, leading to the model being called a densely connected network.

The main motivation for the researchers who came up with the dense network was trying to solve the problem of input or gradient information getting faded away as it passed through many layers in either direction. The paper also noted that there are other models proposed that try to solve this issue such as ResNets and Highway Networks. However, these models apply a common theme, which is to find the shortest way from the early layers to the end layers. And, this is where their design of the DenseNet is meant to maximize the amount of information flow between all the layers [12]. Another major advantage of the DenseNet121 model is that it only uses around 8 million parameters while VGG16 has 138 million. This difference can greatly affect the energy used and time consumed to train a DenseNet model. Apart from just lesser number of parameter one of the major benefits of this architecture is the layers having easy access to the gradients information which can lead to easier training of the model [12].

Furthermore, the Mixed Link Network that was proposed in 2018 discusses that the DenseNet model falls under this larger category called **Dense Topology**. They argue that even though DenseNet is a successful model there is still room for improvements in regards to redundancy where the same type of raw features are passed down different layers. With the Mixed Link Network, they essentially came up with a new breed that combines the best of DenseNet and ResNet while also potentially trying to remove some of their drawbacks in the process[13].

For the DenseNet121 model, we tried 3 main ways to train it. First was to train with no weights, no fine tuning, 150 epochs and early stopping. Next, was to use the ImageNet weights with no fine-tuning and running for 150 epochs. And, finally the third way was to use ImageNet weights, with 150 epochs and fine tuning for 50 epochs. The reason behind this breakdown was to mainly see what happens with and without ImageNet weights. ImageNet dataset is one of the most popular image classification dataset in the Machine Learning community. So, using the weights from that dataset, there is a high likelihood that the model will perform better. But, we truly want to see what else we can learn about it.

3.5. EfficientNetB0

EffeicientNet was first discussed in "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks" in 2019.[14] It proposes that convolution neural networks, which are widely used, can achieve better accuracy through carefully balancing the depth, width, and resolution. This balancing is achieved all through the use of a simple, yet highly effective compound coefficient. The results from implementation of this are extremely good. The EfficientNetB0 model can achieve 84.3% accuracy on ImageNet. It achieves this while also being much faster (6.1x), and smaller (8.4x), than the best ConvNet. Previous works took an approach of scaling up either depth, width, or image resolution, usually arbitrarily, but this approach lead to less than ideal accuracy. The difficulty is that depth, width, and resolution are depend upon each other to determine the optimal scaling, hence why most ConvNets just focus on scale one of the three.

The coding approach was to initially train it with just 10 epochs, both with and without fine tuning, while using ImageNet weights. We then used 100 epochs for the training, and 50 for the fine tuning, also including ImageNet. The early stopping resulted in it stopping around 30 epochs or so, as such we deemed not worth while to increase the epochs any higher. We then trained with the 100 epochs training/50 epochs fine tuning but without the ImageNet weights. Finally we trained with the 100 epochs training/50 epochs fine tuning but added in a drop out layer to see how that would effect performance.

3.6. Inception V3

Originally proposed in the paper, "Rethinking the Inception Architecture for Computer Vision" by Szegedy et. al., the Inception V3 model is the result of many design principles being applied to the Inception architecture [15]. These improvements aimed to leverage the already low computational costs of the Inception model and optimize its performance for larger-scale applications. Due to the relatively generic structure of the Inception model, it was easier to observe the effect that various design principles had on the performance of the model when compared to other competing models from VGGNet or GoogLeNet.

The resulting model was found to perform competitively with other models at the expense of a small increase in the computational costs, but still outperforming many of the denser networks, being six times cheaper in computation costs and five times smaller in the number of parameters used.

Since the original paper, Inception V3 has become a widely-used model for image classification tasks, performing quite well on the ImageNet dataset with an accuracy of 78.1% or greater[16]. Due to its high accuracy and partially due to how Keras is capable of instantiating the model natively through its applications library, Inception V3 was selected as one of the models to be tested in this project.

For this project, the approach used to evaluate the Inception V3 model consisted of a few different steps. To get a baseline of how the model performs without any weights, the first step was to evaluate the model without the influence of the ImageNet weights and train it completely on the image dataset used for this problem. Secondly, the model was tested using the weights from the ImageNet dataset. Finally, due to some overfitting, a third test was performed by adding a dropout layer of 20% to the model. For all three steps, strategies of initial and fine tuning were implemented using early stopping and model checkpoints in conjunction with a large number of epochs. For initial tuning, a learning rate of 10^{-4} was employed, and for fine tuning, a learning rate of 10^{-8} was employed.

4. RESULTS

Overall, what we noticed as a group was that when we applied the ImageNet weights along with fine-tuning of the model, it led to the best resulting algorithm when compared to training the model without pre-loaded weights.

4.1. VGG16

When transfer learning was utilized (using ImageNet weights), the VGG16 model initially performed very well with a training accuracy of 86.33% and a validation accuracy of 75.30%. The test accuracy for this base-model was 69.49% after fine tuning.

Next, when increasing the range of epochs to 50 for model training, we saw a training accuracy of 90.60% and a validation accuracy of 78.10%. The test accuracy for this 35-epoch model was 73.24% after fine tuning.

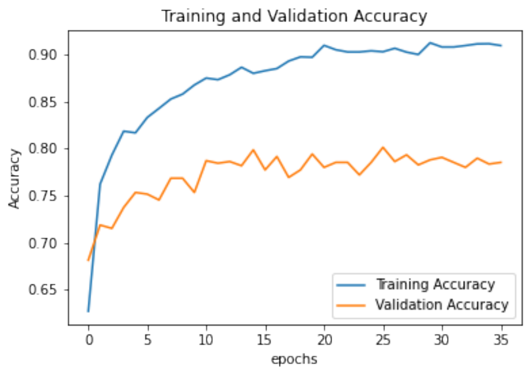

Fig. 4: Train and validation accuracy for VGG16 Transfer Learning after 35 epochs.

Suspecting over-fitting, dropout layer (with a dropout rate of 30%) was added to decrease the different in training and validation scores, and also increase the test score. As seen in the plot below, the addition of the dropout layer resulted in a training accuracy of 86.31% and a validation accuracy of 80.70%. The final testing accuracy after fine tuning was 74.50%.

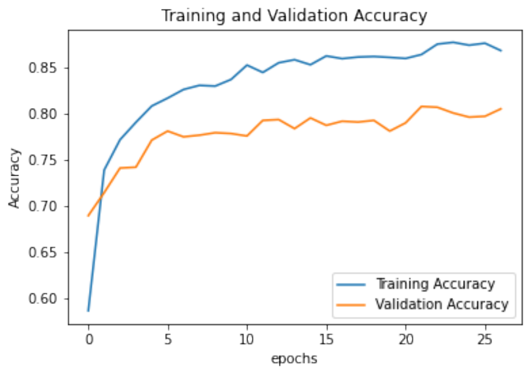

Fig. 5: Train and validation accuracy for VGG16 Transfer Learning with a dropout layer after 26 epochs.

To summarize, the VGG16 model that utilized a transfer learning process with ImageNet weights, a 50 epoch training range, and a dropout layer of 30% performed the best of all 4 attempted models. A test accuracy of 74.50% was achieved after the fine-tuning stage. Compared to the original model (with a test accuracy of 25.77%) where the VG166 model was trained from scratch with no ImageNet weights, it is clear that the added modifications were necessary to increase the overall test accuracy. This makes a strong argument that using transfer learning in conjunction with a VGG16 model is a valid and powerful method to classify garbage images.

4.2. ResNet50

The ResNet50 neural network yielded relatively strong results throughout training and validation with most models once transfer learning was employed. For the initial model without transfer learning the training and validation accuracy scores were 63.49% and 56.46%. Adding transfer learning on the model using ImageNet weights with 10 epochs immediately yielded significantly better results at 94.43% training, 81.92% validation accuracy. Once expanded to 100 epochs, although early stopping was reached at only 12 epochs, the results slightly improved in validation to 93.91% training and 83.70% validation accuracy. Due to that gap between training set accuracy scores, an indication of overfitting, when 20% dropout was added the model slightly improved again to 95.87% training and 84.15% validation accuracy. Additionally 50% dropout was also briefly tested but yielded slightly worse results so the final chosen model was with 20% dropout. For this model the 22 epoch accuracy history can see below.

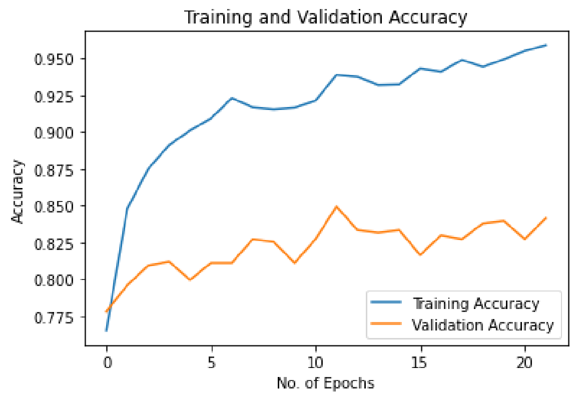

Fig. 6: ResNet50 transfer learning with ImageNet weights and 20% dropout.

Using the final ResNet50 model which included transfer learning with ImageNet weights, and a 20% dropout layer, using the isolated test set a test accuracy of 81.28% was attained after fine tuning the model, with a cross-entropy loss of 0.9041. To add some context evaluating the initial ResNet50 model, which had no transfer learning, a testing score of only 56.75% was attained, with a cross-entropy loss of 1.2538. Clearly using a transfer learning approach to garbage classification is the correct choice for the ResNet50 model.

4.3. DenseNet121

Firstly, we will discuss the results of the DenseNet121 model when no pre-loaded weights are added and no fine-tuning is done. We achieved a training accuracy of just under 70% while validation accuracy of just under 60%. When we applied this model to the test set and evaluate it, we got quite a poor accuracy of 52.66%. This goes to show that training from scratch was certainly a critical factor in the poor results, apart from the fact that the model ran for 90 epochs before early stopping. 90 epochs is a significant number of iterations to only achieve the said accuracy.

Next, we look at the results for when we applied the ImageNet weights with no fine-tuning of the model for 150 epochs. In Figure 7, we can see the training and validation accuracy graph and it shows that the model stopped training after 15 epochs. This is due to the fact that the code incorporates early stopping. What is also interesting to see here is that at the beginning, the accuracy of the model is not that great and just within 15 epochs, the training accuracy goes up by 20% and the validation accuracy goes up by 10%. What is also important to note here is that the model achieved this result with just 15 epochs compared to the case without using ImageNet weights, where the model trained for 90 Epochs. Those extra epochs mean more usage of energy and time to train the model. And, it is not a minor difference, 90 epochs is 6 times as many as when using the ImageNet weights.

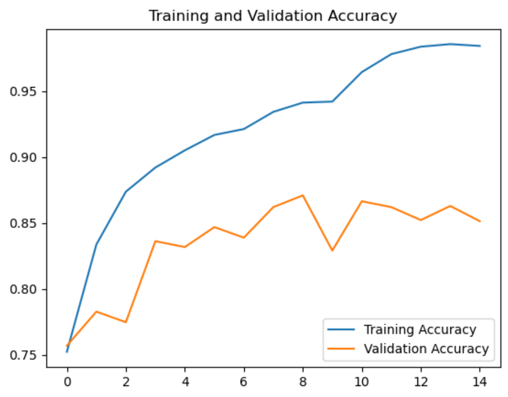

Fig. 7: DenseNet121 - ImageNet weights, No Fine-Tuning

Finally, we will discuss the results for when we apply the ImageNet weights along with fine-tuning of the model. This approach gave us the best result. As seen in Figure 8, the Validation accuracy is at 86% while the overall Training accuracy is just under 96%. Figure 9 represents the Validation and Training accuracy for the Fine-Tuning step of the model. Finally, we applied the classifier's weights to the Test set and evaluated the model. For the initially trained model, we got an accuracy of 81.66% and with fine-tuned model we got an accuracy of 82.15%.

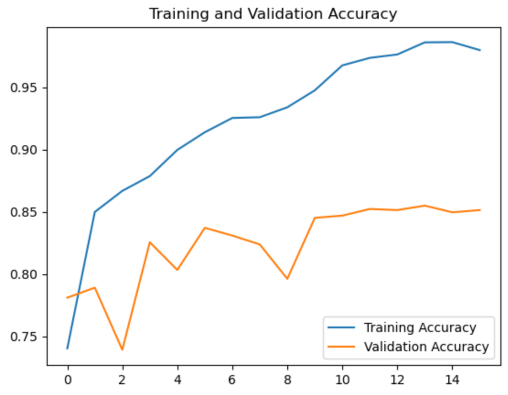

Fig. 8: DenseNet121 - ImageNet weights (Initial Training)

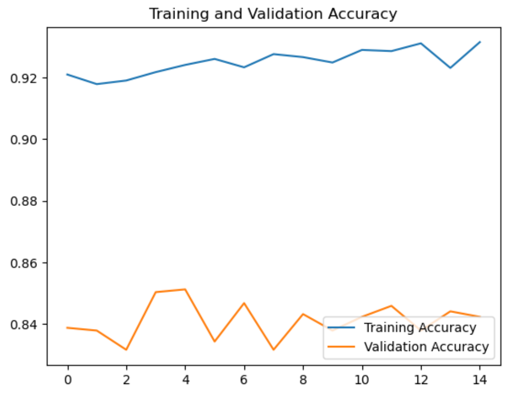

Fig. 9: DenseNet121 - ImageNet weights, Fine-Tuning

4.4. EfficientNetB0

We will now discuss the results of the EfficientNetB0 model. First is the case with only 10 epochs and no fine tuning. We achieved a training accuracy of 92.78%, a validation accuracy of 80.32%, and a test score of 76.08%. Adding fine tuning with 5 epochs we get a slight decrease in the training accuracy, but an increase in the validation and testing accuracy. We achieved a training accuracy of 90.36%, a validation accuracy of 80.59%, and a test score of 78.81%.

Next we will look at the case with 100 epochs. We utilized early stopping and our model finished after 23 epochs. We achieved a training accuracy of 95.31%, a validation accuracy of 84.24%, and a test score of 80.17%. Adding fine tuning with 50 epochs we get a slight improvement to the accuracy. We achieved a training accuracy of 94.18%, a validation accuracy of 83.35%, and a test score of 79.68%.

Next we tried the model without ImageNet just to see how not utilizing transfer learning effects the accuracy. We found that we go very bad results when not using ImageNet. We achieved a training accuracy of 26.20%, a validation accuracy of 25.73%, and a test score of 25.53%. Adding fine tuning with 50 epochs we get a slight improvement to the training accuracy. We achieved a training accuracy of 26.38%, a validation accuracy of 25.73%, and a test score of 25.53%.

From the previous results, we can see that the model was likely over-fitting since there is a large difference between the training and validation scores. In an effort to combat the effects of over-fitting, the third and final training step of the model was to add a 20% dropout layer. As shown in Fig. 10 we achieved a training accuracy of 95.31%, a validation accuracy of 84.24%, and a test score of 80.30%. Adding fine tuning with 50 epochs we get a slight improvement to the test accuracy. As shown in Fig. 11 we achieved a training accuracy of 94.18%, a validation accuracy of 83.35%, and a test score of 81.41%.

With the drop out layer we get no improvement to the training or validation scores, but a slight increase in the test scores, so the dropout layer did help make some marginal improvements.

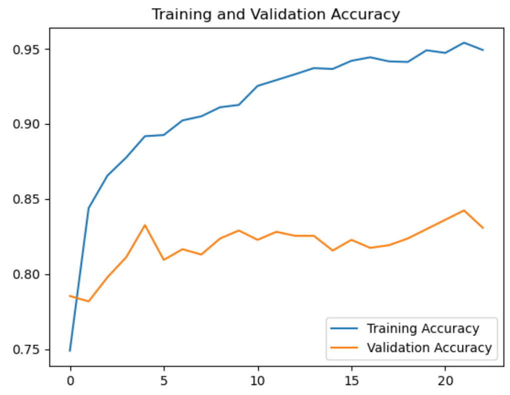

Fig. 10: EfficientNetB0 transfer learning with ImageNet weights and 20% dropout. (Initial Training)

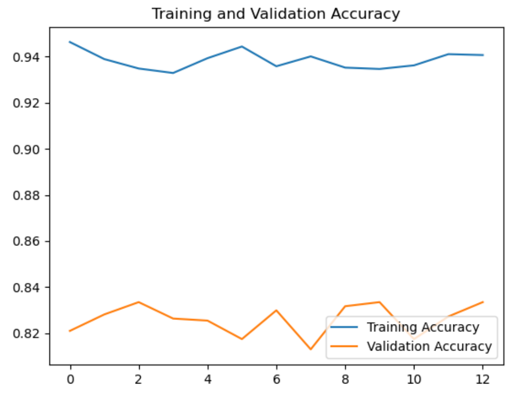

Fig. 11: EfficientNetB0 transfer learning with ImageNet weights and 20% dropout. (Fine Tuning)

4.5. InceptionV3

For the Inception V3 model, early stopping and model checkpoints were employed alongside 1000 epochs to ensure that the model would converge on a final result.

The initial training step for the Inception V3 model without the weights of the ImageNet dataset resulted in a model that achieved an accuracy of 74.65% on the training set. On the validation set, the same model achieved an accuracy of 66.43% with a cross-entropy loss of 1.1926. With a fine tuning learning rate of 10^{-8}, this model converged on an training accuracy of 74.06%, a validation accuracy of 66.16%, and a cross-entropy validation loss of 1.1956.

When evaluated against the test set, the model performed worse overall, with an accuracy of 53.90% for the initial tuning and an accuracy of 54.40% for the fine-tuning. The cross-entropy losses of the models were 1.2571 and 1.2575, respectively.

With a baseline established, the next step consisted of adding the weights from the ImageNet dataset to the model. The resulting Inception V3 model achieved a training accuracy of 93.68%, a validation accuracy of 81.66% and a cross-entropy validation loss of 0.5280. Upon fine-tuning, the model converged on a training accuracy of 92.82%, a validation accuracy of 82.99%, and a cross-entropy validation loss of 0.5073.

When evaluated against the test set, the model with the ImageNet weights resulted in an accuracy of 76.70% and 79.56% for the initial tuning and fine-tuning, respectively, along with cross-entropy losses of 0.6401 and 0.7064.

Figures 12 and 13 showcase the accuracies on the training and validation sets for the model with and without fine-tuning.

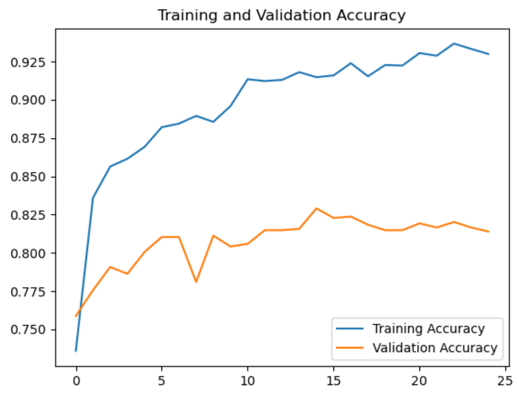

Fig. 12: Inception V3, ImageNet Weights

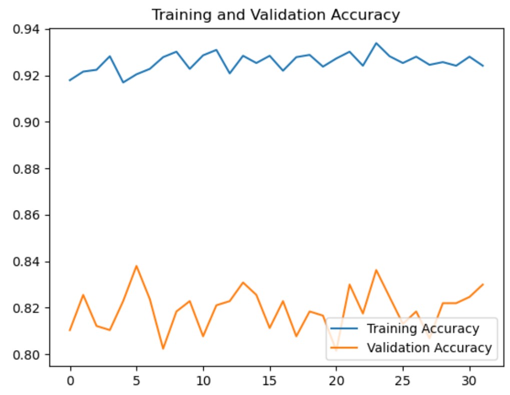

Fig. 13: Inception V3, ImageNet Weights, Fine-Tuning

From these results, it was observed that the model was likely overfitting. In an effort to combat the effects of overfitting, the third and final training step of the Inception V3 model was to add a 20% dropout layer. The initial tuning of this model had a training accuracy of 93.24%, validation accuracy of 82.90%, and a cross-entropy validation loss of 0.5248. Upon fine-tuning, these numbers were adjusted to 92.48%, 83.08%, and 0.5123, respectively.

The results on the test set saw small changes, with accuracies of 77.45% and 77.82%, and cross-entropy losses of 0.6249 and 0.6077 for the initial and fine-tuning models. Overall, this model performed slightly worse than the base ImageNet model without dropout.

5. DISCUSSION

The results of the best-performing cases for each of the models we tested can be seen in Table 1. In general, it was found that transfer learning with ImageNet weights had a major improvement on the accuracy of the model. This makes sense due the large size of the ImageNet dataset, which has millions of photos that are used in the training of the models. In comparison, the dataset used in this project only had a few thousand images, which is considerably less than the ImageNet dataset. Therefore, when training the models without the weights from the ImageNet set, it would have less data to learn features from. Another to note is that by changing the learning rate of the model without using Imagnet weights, it had almost no impact on the performance of the models.

Adding a drop-out layer was found to have inconsistent results, increasing the accuracies for the VGG16, ResNet50, and EfficientNetB0 but having decreased accuracies for other models, such as the Inception V3.

The best-performing model on the test data of the five developed in this project was the DenseNet121 model, with an accuracy of 82.15%. Many of the remaining models performed considerably well, trailing behind DenseNet's performance by only 1-2%. The exception to this is the VGG16 model which performed with an accuracy of 74.5%.

Table 1: Model Performance on Test Data

Model	**Description**	**Test Accuracy**
VGG16	With ImageNet, Drop-Out, Fine-Tuning	0.745
ResNet50	With ImageNet, Drop-Out, Fine-Tuning	0.813
DenseNet121	With ImageNet, Fine-Tuning	0.82
EfficientNetB0	With ImageNet, Drop-Out, Fine-Tuning	0.814
InceptionV3	With ImageNet, Fine-Tuning	0.796

6. CONCLUSION

This project covered the popular topic of garbage classification, an important problem with real-world implications that can be applied to daily life. To accomplish this task, each member of the group chose a model from the Keras library and set out to train a model that would perform well on a garbage dataset provided by Dr. Souza.

Out of the five models trained for this project, DenseNet121 was found to perform the best on the test set with an accuracy of 82.15%, although all models performed well with accuracies of 74.5% or greater. It was generally found that ImageNet weights had a significant impact on the model accuracy, due to its large and varied dataset.

An accuracy of 82.15% is speculated to be quite satisfactory for the purposes of garbage classification, as some people are likely to misclassify garbage at a rate worse than 82.15%. Such an algorithm could prove useful for the purposes of developing an application that classifies garbage, or for companies to correctly separate their waste into their proper categories and help curb the amount of incorrect garbage contributing to landfill waste.

7. REFERENCES

[1] E. Sybingco E. Dadios E. Calilung S. Rabano, M. Cabatuan, “Common garbage classification using mobilenet,” in *2018 IEEE 10th International Conference on Humanoid, Nanotechnology, Information Technology,Communication and Control, Environment and Management (HNICEM)*, 2018, pp. 1–4.

[2] T. Xiao J. Gao J. Bai W. Luo B. Dong S. Chen, J. Huang, “Carbon emissions under different domestic waste treatment modes induced by garbage classification: Case study in pilot communities in shanghai, china,” *Science of The Total Environment*, vol. 717, pp. 137–193, 2020.

[3] “Keras applications,” Keras. [Online] keras.io/api/applications//, Accessed: March 16, 2022.

[4] K. Simonyan A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *Journal*, September 2014.

[5] R. Thakur, “Step by step vgg16 implementation in keras for beginners,” Towards Data Science. [Online] https://medium.com/towards-data-science/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c, Accessed: Mar 27, 2022.

[6] P. Varshney, “Vggnet-16 architecture: A complete guide,” Kaggle. [Online] https://www.kaggle.com/code/blurredmachine/vggnet-16-architecture-a-complete-guide/notebook, Accessed: Mar 31, 2022.

[7] S. Ren J. Sun K. He, X. Zhang, “Deep residual learning for image recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR*, 2016, pp. pp. 770–778.

[8] Great Learning Team, “Introduction to resnet or residual network,” Great Learning. [Online] www.mygreatlearning.com/blog/resnet/, Sep 28, 2020.

[9] A. Kaushik, “Understanding resnet50 architecture,” OpenGenus IQ. [Online] iq.opengenus.org/resnet50-architecture/, Accessed: Mar 24, 2022.

[10] F. Chollet, “Transfer learning fine-tuning,” Keras. [Online] https://keras.io/guides/transfer_learning/, May 12, 2020.

[11] P. Sadowski P. Baldi, “Understanding dropout,” in *Advances in Neural Information Processing Systems*, M. Welling Z. Ghahramani K. Q. Weinberger C. J. C. Burges, L. Bottou, Ed. 2013, vol. 26, Curran Associates, Inc.

[12] L. van der Maaten K.Q. Weinberger G. Huang, Z. Liu, “Densely connected convolutional networks,” *Journal*, January 2018.

[13] W. Wang J. Yang X. Li, T. Lu, “Mixed link networks,” *Journal*, February 2018.

[14] Q.V. Le M. Tan, “Efficientnet: Rethinking model scaling for convolutional neural networks,” *Arxiv*, January 2019.

[15] S. Ioffe J. Schlens Z. Wojna C. Szegedy, V. Vanhoucke, “Rethinking the inception architecture for computer vision,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[16] Google Cloud, “Advanced guide to inception v3,” Google Cloud. [Online] https://cloud.google.com/tpu/docs/inception-v3-advanced, Accessed: Mar 23, 2022.