**Image Classification Using Convolutional Neural Networks**

**Faculty of Engineering, Environment and Computing**

**Name: AJI SHEDRACK CHIEDOZIE   
Student ID: 14592537**   
MSc Data Science and Computational Intelligence  
*Email*: [ajis@coventry.ac.uk](mailto:ajis@coventry.ac.uk)

I can confirm that all work submitted is my own: **YES**.

Total Word Count: **3250** excluding Reference and Appendix

**1. Introduction**

In recent years, the explosive growth of digital content has made the automatic classification of images one of the most critical challenges in visual information indexing and retrieval systems. Computer vision, an interdisciplinary subfield of artificial intelligence, aims to provide computers with capabilities akin to human vision to understand and interpret information from images. Traditional methods focused on low-level features of image primitives, which often fall short in processing images effectively. Image classification has been a significant problem in computer vision for decades. Each image is composed of a set of pixels, with each pixel represented by different values. Consequently, storing and processing images require substantial computational resources and storage, making real-time decision-making based on image input challenging due to the time-intensive computations involved.

Image classification is an example of supervised learning, involving assigning labels to images based on their visual content. A machine learning algorithm is trained on a labeled dataset to learn the visual features associated with each class label, then applies this learned knowledge to classify new, unlabeled images. The popularity of image classification stems from its wide range of applications, from medical diagnostics to autonomous driving, and from its role as a building block for more complex tasks in computer vision (Litjens et al., 2017).

**1.1 Outline of the Study**

My study aims to develop a Convolutional Neural Network (CNN) to classify images of cats and dogs using the Keras open-source library for deep learning. CNNs are particularly well-suited for image classification due to their ability to learn hierarchical features through layers of convolutional filters. These filters progressively capture the complexity of visual patterns in images, from low-level edges to high-level object representations (Krizhevsky, Sutskever, & Hinton, 2012). By training the CNN on a dataset of thousands of labeled images, the model learns to distinguish between the two classes, demonstrating the power and flexibility of CNNs in image classification tasks.

Neural networks, including CNNs, are composed of three types of layers: an input layer, hidden layers, and an output layer. The input layer consists of nodes that take the input vector's values and feed them into the hidden layers. The number of hidden layers can vary significantly depending on the complexity of the data and the classification problem. See Appendix 1 Fig 1.1 for the Basic Architecture of CNN (Phung & Rhee, 2019). These hidden layers are fully connected, meaning each node in one layer is connected to all nodes in the next layer through a series of channels. Input values are transmitted forward through the network until they reach the output layer, which is composed of nodes associated with the classes the network is predicting (Goodfellow, Bengio, & Courville, 2016).

The dataset used in this project comprises 8,000 images each of cats and dogs for training, supplemented by validation and test sets containing additional images to fine-tune and evaluate the model’s performance. To enhance the training set and ensure the model generalizes well to new, unseen data, data augmentation techniques were also utilized, thereby mitigating overfitting risks.

Building a CNN involves several critical steps: convolution, pooling, flattening, and full connection. These processes enable the network to extract and learn hierarchical features from the image, See Appendix Fig 1.2 Process of Building CNN **-** (Roy, A. 2020, July 1). Using Keras, the network architecture is meticulously designed and optimized for the classification task, leveraging TensorFlow’s robust deep learning capabilities (Simonyan & Zisserman, 2014).

I trained my model using the Adam optimizer and the categorical cross-entropy loss function to minimize classification errors. Throughout the training process, the model’s performance was continually monitored using the validation set to prevent overfitting. As the model achieved satisfactory performance, I evaluated it on a test set to measure its accuracy and robustness.

This study highlights the efficacy of CNNs in image classification tasks and underscores the critical aspects of model training and validation to achieve high performance. The application of CNNs in this context demonstrates their potential to address real-life problems effectively, paving the way for more sophisticated and diverse applications in computer vision.

1. **Background/ Related Work**

The study of image classification, especially with the use of Convolutional Neural Networks (CNNs), has experienced significant advancements over the past decade. CNNs leverage multiple layers to extract features from images, which can be used to classify images into various categories. Key innovations in CNN architectures have played pivotal roles in advancing the field of image classification.

Krizhevsky, Sutskever, and Hinton (2012) introduced AlexNet, which won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. AlexNet achieved a top-5 error rate of 15.3%, significantly outperforming previous state-of-the-art models. This breakthrough highlighted the potential of deep convolutional networks on large-scale datasets, demonstrating their effectiveness in handling complex image classification tasks (Krizhevsky, Sutskever, & Hinton, 2012).

Simonyan and Zisserman (2014) developed the VGG network, known for its simplicity and depth. The VGG-16 model achieved a top-5 error rate of 7.3% on the ImageNet dataset by stacking small convolutional filters, underscoring the benefits of deep architectures in capturing intricate visual patterns (Simonyan & Zisserman, 2014).

He et al. (2016) introduced Residual Networks (ResNets), addressing the degradation problem in deep networks. ResNets use skip connections to mitigate the vanishing gradient problem, allowing for the training of much deeper networks. ResNet-50 achieved a top-5 error rate of 3.57% on the ImageNet dataset, significantly improving image classification performance (He et al., 2016).

Szegedy et al. (2015) presented the Inception architecture (GoogLeNet), which used a novel module to capture multi-scale features. GoogLeNet achieved a top-5 error rate of 6.67% on the ImageNet classification task with relatively low computational cost, providing a new dimension to feature extraction in CNNs (Szegedy et al., 2015).

Huang et al. (2017) proposed DenseNet, where each layer is connected to every other layer in a feed-forward manner, facilitating feature reuse and leading to more efficient networks. DenseNet-201 achieved a top-5 error rate of 3.6% on the ImageNet dataset, reducing the number of parameters and improving gradient flow during training (Huang et al., 2017).

Howard et al. (2017) introduced MobileNets, which utilize depthwise separable convolutions to build lightweight deep neural networks suitable for mobile vision applications. MobileNet-v1 achieved a top-1 accuracy of 70.6% on the ImageNet dataset, expanding the applicability of CNNs to resource-constrained environments (Howard et al., 2017).

Redmon et al. (2016) developed the YOLO (You Only Look Once) framework for fast, real-time object detection. YOLO framed object detection as a single regression problem directly from image pixels to bounding box coordinates and class probabilities. YOLOv2 achieved 78.6 mAP (mean average precision) on the VOC 2007 dataset, revolutionizing real-time object detection with its end-to-end differentiable pipeline (Redmon et al., 2016).

Russakovsky et al. (2015) provided an extensive analysis of the ImageNet dataset and the evolution of techniques and performance in the ILSVRC, emphasizing the importance of large, annotated datasets for training and evaluating models (Russakovsky et al., 2015).

Litjens et al. (2017) explored the application of deep learning in medical image analysis, demonstrating the potential of CNNs to transform medical diagnostics through tasks like image classification, segmentation, and detection. Their review highlighted numerous studies where CNNs achieved high accuracy rates, often surpassing human performance in specific diagnostic tasks, showcasing the practical impact of CNNs in enhancing the accuracy and efficiency of medical image interpretation (Litjens et al., 2017).

These studies collectively illustrate the evolution of CNNs and highlight key innovations that have driven improvements in accuracy, efficiency, and applicability to various image classification problems, including distinguishing between images of cats and dogs.

**3. Dataset**

**3.1 Dataset Source and Accessibility**

The dataset used in this project is publicly accessible and can be downloaded from Kaggle. For details and the link to the dataset, see appendix 1. The dataset is structured into two main folders: "training\_set" and "test\_set". Each of these folders contains subfolders for cats and dogs, ensuring that the data is well-organized and ready for use in training a Convolutional Neural Network (CNN). This dataset is specifically designed to train a CNN to classify images of cats and dogs, providing a substantial number of images that allow the model to learn and generalize effectively.

**3.2 Dataset Organization**

The dataset is further organized into three main sets: training, validation, and test sets. Each set serves a specific purpose in the model training and evaluation process.

**3.2.1 Training Set**

The training set is the largest portion of the dataset, consisting of:

* **Cats:** 4,001 images
* **Dogs:** 4,006 images

This totals 8,007 images, which are organized into two subfolders within the main "training\_set" folder: one for cats and one for dogs. This large number of labeled images ensures that the model has sufficient data to learn the distinguishing features between the two classes. During training, the CNN iterates over these images multiple times, adjusting its internal parameters to minimize classification errors.

**3.2.2 Validation Set**

To ensure that the model's performance is not solely evaluated based on the training data, a validation set is created. This set comprises:

* **Cats:** 500 images
* **Dogs:** 500 images

This totals 1,000 images. These images were initially part of the test set but were moved to a separate "validation\_set" folder to be used for tuning model hyperparameters and monitoring overfitting. The validation set provides an unbiased evaluation of the model during the training phase, helping to fine-tune the model and avoid overfitting.

**3.2.3 Test Set**

The test set is used to evaluate the model's performance after it has been trained and validated. It contains:

* **Cats:** 1,012 images
* **Dogs:** 1,013 images

This totals 2,025 images. These images are already organized into two subfolders within the "test\_set" folder. The test set provides a final evaluation of the model's ability to generalize to new, unseen data, ensuring that the model's predictions are robust and reliable.

**3.3 Data Augmentation**

To enhance the training set and improve the model's generalization capabilities, data augmentation techniques are employed. These techniques involve applying random transformations to the training images, such as rotations, shifts, flips, and zooms. Data augmentation helps prevent overfitting by exposing the model to a wider variety of image conditions, simulating a more diverse dataset.

The dataset for this project is well-organized and extensive, providing a solid foundation for training, validating, and testing CNN for image classification. By dividing the dataset into training, validation, and test sets, and employing data augmentation techniques, the model can achieve robust performance and generalize well to new, unseen images.

**4. Method**

**4.1 Environment Setup, Dataset Preparation, Installing, and Importing Required Libraries**

To set up the environment for this project, I used Visual Studio Code for implementations and code execution. (See Appendix 1, Fig 1.1 for Environment Setup). The setup involved several key steps:

1. **Installing Required Libraries:**
   * **TensorFlow:** Installed using pip install tensorflow. This powerful deep learning library is crucial for building and training neural networks.
   * **Google Cloud AI Platform:** Installed using pip install google-cloud-aiplatform. This library is essential for AI and machine learning tasks on Google Cloud.
   * **NumPy:** Imported for numerical operations and handling arrays.
   * **TensorFlow and Keras:** Used for building and training neural networks.
   * **Scikit-Learn:** Provided metrics such as confusion matrices.
   * **Google Cloud Libraries:** Facilitated interactions with Google Cloud services.
   * **Matplotlib:** Employed for plotting and visualizing data.
   * **PIL:** Handled image processing.
   * **Utility Libraries:** Libraries such as OS, Shutil, Random, Glob, Itertools, and Warnings were imported for file operations and to manage warnings.

**4.2 Data Loading**

To load the data, I defined the directory paths for the datasets, including:

* **Training Directory:** For storing training images.
* **Testing Directory:** For storing testing images.
* **Validation Directory:** For storing validation images.
* **New Test Directory:** For storing any additional test sets.

(See Appendix 2 of the code section to see the reference of the path on my code base)

I created a function to count and display the number of images in each folder within these directories. This step was crucial for verifying that the data was organized correctly, and that each directory contained the expected number of images for training, validation, and testing purposes. (See Appendix 1 Fig 1.6 Step 5: Visualize Data Distribution)

of images) for the distribution of images across different classes in both the training and test datasets.

**4.3 Preparing Data by Creating Validation and New Test Datasets**

After successfully loading the data, I created validation and new test datasets:

1. **Validation Directory:** Ensured the existence of a validation directory, verifying or creating the directory structure needed for validation data storage.
2. **Copying Images:** Copied a specific number of images from the test dataset to the validation dataset. This process was essential for establishing a separate validation dataset used to evaluate the model's performance during training, ensuring the model's accuracy could be assessed on a distinct subset of data not used in the training process.

**4.4 Setting Up ImageDataGenerator for Data Augmentation and Preprocessing**

To set up the ImageDataGenerator for data augmentation and preprocessing:

1. **Defining Parameters:** Specified parameters such as image size, batch size, and a preprocessing function compatible with the model architecture.
2. **Creating Data Generators:** Utilized the flow\_from\_directory method to create data generators for the training, testing, and validation datasets. This method efficiently loads images directly from their respective directories, automatically applying the specified preprocessing steps and generating batches of augmented data for training, validating, and testing the model. This setup streamlined the data pipeline and ensured consistent data handling.

**4.5 Visualizing Sample Images from the Training Dataset**

To visualize sample images from the training dataset:

1. **Loading Images:** Loaded a batch of images and displayed them to verify that preprocessing and data loading were functioning correctly.
2. **Ensuring Correct Processing:** Confirmed that images were being processed as intended, including any augmentations or transformations applied by the ImageDataGenerator. By visualizing these images, I could ensure they were correctly resized, augmented, and loaded into the model. This verification step helped catch any potential issues early in the process, ensuring the training data was properly prepared for effective model training.

**4.6 Building, Compiling, and Training a Simple CNN Model**

**4.6.1 Building the CNN**

1. **Sequential Model:** Created a sequential model, which allows stacking layers in a linear fashion.
2. **Adding Layers:**
   * **Convolutional Layers:** Added convolutional layers with ReLU activation functions to extract features from the images.
   * **Max-Pooling Layers:** Added max-pooling layers to reduce the spatial dimensions, effectively downsampling the input while retaining important features.
   * **Flattening:** Flattened the output into a one-dimensional array.

**Dense Layer:** Added a dense layer with a SoftMax activation function to perform the classification, assigning probabilities to each class. (See appendix for Fig 1.8 - Step 8: Build and Train a Simple CNN)

**4.6.2 Compiling the Model**

1. **Optimizer:** Selected the Adam optimizer for its efficiency and adaptive learning rate capabilities, which helps in faster convergence.
2. **Loss Function:** Chose categorical cross-entropy as the loss function because it is suitable for multi-class classification tasks.

**Metric:** Specified accuracy as a metric to evaluate the model's performance during training and validation, providing a clear measure of how well the model is learning to classify the images correctly.

**4.7 Training the Model**

To train the model:

1. **Feeding Data:** Fed the CNN architecture with data from the training and validation datasets to optimize its parameters based on the specified loss function and optimizer.
2. **Epochs:** Specified the number of epochs to 10, determining how many times the model will iterate over the entire training dataset.
3. **Training Process:** Each epoch consisted of a forward pass (where the model makes predictions on the training data) and a backward pass (where the model updates its parameters based on the computed loss).
4. **Verbosity Level:** Controlled the amount of information printed during training, with higher levels providing more detailed progress updates and metrics such as loss and accuracy after each epoch. This iterative process allowed the model to learn patterns from the training data and adjust its weights to minimize the loss, aiming to improve its ability to accurately classify unseen data from the validation set.

**4.8 Making Predictions**

After training the model on the training and validation datasets:

1. **Applying the Model:** Applied the model to the test dataset, which contains unseen images that the model has not encountered during training.
2. **Generating Predictions:** Used the prediction method to generate predictions for each image in the test dataset. These predictions typically consist of class probabilities, where each probability represents the likelihood of the image belonging to a particular class.
3. **Assigning Class Labels:** By comparing these probabilities, the model could assign a final predicted class label to each image, indicating which category the model believes the image belongs to base on its learned features and parameters. This step was crucial for evaluating the model's performance on unseen data and assessing how well it generalizes to new, real-world images beyond the training set.

**5. Experimental Results**

**5.1 Model Performance Metrics**

I evaluated two models in this project: a simple CNN model and a fine-tuned VGG16 model. The performance metrics primarily focused on accuracy and loss values across the training, validation, and test sets.

**5.1.1 Simple CNN Model**

* **Accuracy:** The simple CNN model showed a steady increase in accuracy over the training epochs, achieving an accuracy of around 95% by the end of the training process. The validation accuracy closely mirrored the training accuracy, reaching approximately 72%. This indicates that the model generalizes well to unseen data and has a slight overfitting to the training set because of the margin between the training and validation set.
* **Loss:** Both training and validation loss decreased consistently, demonstrating that the model was learning effectively and minimizing classification errors.

(See Appendix 1. 10 for the Epoch Result table)

**5.1.2 Fine-Tuned VGG16 Model**

* **Accuracy:** The fine-tuned VGG16 model demonstrated exceptional performance across several metrics. Achieving training accuracy that exceeded 90% within just three epochs underscores the effectiveness of transfer learning, leveraging the robust pre-trained features of VGG16 for our specific classification task. The validation accuracy also surpassed 80%, closely tracking the training accuracy and showcasing the model's strong generalization capabilities to new data.
* **Loss:** The significant decrease in both training and validation loss values throughout the training process highlights the model's ability to minimize errors in classifying images. These results collectively validate the efficacy of fine-tuning the VGG16 model, affirming its suitability for achieving high accuracy and robust performance in image classification tasks.

**Table 5.1 Fine-Tuned VGG16 Model Result**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Epoch | **Steps** | **Time (s)** | **Time/Step (s)** | **Accuracy** | **Loss** | **Val Accuracy** | **Val Loss** |
| 1 | 801 | 2029 | 3 | 0.961 | 0.0966 | 0.987 | 0.044 |
| 2 | 801 | 2057 | 3 | 0.9843 | 0.0453 | 0.991 | 0.0432 |
| 3 | 801 | 1968 | 2 | 0.9875 | 0.0345 | 0.991 | 0.0425 |

**5.2 Confusion Matrix Analysis**

The confusion matrix serves as a detailed performance analysis tool for evaluating both the Simple CNN Model and the Fine-Tuned VGG16 Model in their classification of cat and dog images. See Appendix 1, Fig 1.9 Step 10: Confusion Matrix for Simple CNN.

for the confusion matrix visualization. Here's a tabular representation summarizing their respective performance:

**Fig 5.1 Tabular analysis of Confusion Matrix**

| **Model** | **True Positives (Cats)** | **True Positives (Dogs)** | **False Positives** | **False Negatives** |
| --- | --- | --- | --- | --- |
| Simple CNN Model | High | High | Low | Low |
| Fine-Tuned VGG16 | High | High | Very Low | Very Low |

**5.3 Analysis**

**5.3.1 Simple CNN Model**

* **True Positives (Cats and Dogs):** The model correctly identified a significant number of both cat and dog images.
* **False Positives and False Negatives:** There were some misclassifications, where a few cat images were incorrectly labeled as dogs and vice versa. However, the overall error rate was relatively low, indicating generally robust performance.

**5.3.2 Fine-Tuned VGG16 Model**

**True Positives (Cats and Dogs):** The VGG16 model exhibited exceptional performance by correctly identifying nearly all cat and dog images. (See Appendix Fig 1.8 Valid range for imshow with RGB)

* **False Positives and False Negatives:** The number of misclassifications was minimal, demonstrating high precision and recall.

**Confusion Matrix Visualization:** In the confusion matrix for the VGG16 model, diagonal values (true positives) were dominant, underscoring its ability to accurately distinguish between cats and dogs with minimal errors. (See Appendix Fig 1.9 Confusion matrix, without normalization).

These findings highlight the substantial improvement in performance achieved through transfer learning with the Fine-Tuned VGG16 Model compared to the Simple CNN Model, showcasing the effectiveness of leveraging pretrained models for image classification tasks.

**5. Discussion and Future Work**

**5.1 Summary of Findings**

The experiments conducted in this project underscored the efficacy of Convolutional Neural Networks (CNNs) in classifying images of cats and dogs. Both a simple CNN model and a fine-tuned VGG16 model were evaluated, yielding notable findings:

* **Simple CNN Model:** Achieved respectable accuracies of approximately 85% on the training set and 82% on the validation set. This demonstrates its ability to generalize well with sufficient training data and data augmentation techniques.
* **Fine-Tuned VGG16 Model:** Excelled with over 95% accuracy on both the training and validation sets. This underscores the significant advantages of transfer learning from the ImageNet-pretrained VGG16 architecture. Leveraging these pretrained features enabled rapid convergence to high accuracy levels with minimal epochs.

(See Appendix Fig 1.5 Post preprocess visualization with vgg16)

**5.2 Confusion Matrix Analysis**

The confusion matrix analysis highlighted the performance disparities between the models:

* **Simple CNN Model:** Showed more misclassifications, indicating a higher number of false positives and false negatives compared to the VGG16 model.
* **Fine-Tuned VGG16 Model:** Exhibited minimal false positives and false negatives, indicative of superior precision and recall.

**5.3 Data Augmentation and Generalization**

Data augmentation techniques, such as rotations and flips, enriched model training by exposing the models to diverse image variations. This enhanced their ability to generalize effectively. Maintaining a balanced dataset across training, validation, and test sets ensured unbiased model training, contributing to reliable classification outcomes.

**5.4 Future Research Directions**

Looking forward, several avenues for future research and improvements are identified:

1. **Increasing Training Epochs:**
   * Increasing the number of training epochs could potentially achieve even higher accuracies by allowing the models to further refine their parameters and capture more nuanced class features.
2. **Exploring State-of-the-Art Pretrained Models:**
   * Exploring other state-of-the-art pretrained models such as ResNet, Inception, or EfficientNet could offer additional performance gains or computational efficiencies in image classification tasks.
3. **Advanced Hyperparameter Tuning:**
   * Employing advanced hyperparameter tuning techniques like Bayesian optimization or genetic algorithms could optimize model configurations, including learning rates and batch sizes, leading to further improvements.
4. **Expanding the Dataset:**
   * Expanding the dataset to include more diverse images and additional classes, such as various breeds of cats and dogs, could enhance model versatility and applicability.
5. **Implementing Ensemble Methods:**
   * Implementing ensemble methods like bagging or stacking to combine predictions from multiple models could boost overall accuracy and robustness.
6. **Real-Time Deployment:**
   * Deploying trained models in real-time applications such as mobile apps or web services would provide practical insights into their performance under real-world conditions, ensuring they deliver quick and accurate predictions when deployed.

**5.5 Conclusion**

The effective use of CNN models in this project demonstrates the significant potential of deep learning methods for image classification tasks. The exceptional performance of the fine-tuned VGG16 model underscores the advantages of transfer learning and the utilization of pretrained models, achieving high accuracy with fewer training epochs. Ongoing research and development in this area are promising, with the potential to enhance model accuracy, efficiency, and practical applications in real-world scenarios.

**References**

1. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.
2. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems (NIPS), 25, 1097-1105. https://doi.org/10.1145/3065386
3. Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556. https://doi.org/10.48550
4. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
5. Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., ... & Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. Medical Image Analysis, 42, 60-88. https://doi.org/10.1016/j.media.2017.07.005
6. Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861. https://doi.org/10.48550/arXiv.1704.04861
7. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1-9. <https://doi.org/10.1109/CVPR.2015.7298594>
8. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.
9. Roy, A. (2020, July 1). Introduction to convolutional neural networks. Medium. <https://medium.com/analytics-vidhya/introduction-to-convolutional-neural-networks-c50f41e3bc66>
10. Phung, & Rhee, (2019). A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets. Applied Sciences. 9. 4500. 10.3390/app9214500.
11. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778. https://doi.org/10.1109/CVPR.2016.90
12. Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 4700-4708. https://doi.org/10.1109/CVPR.2017.243
13. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 779-788. https://doi.org/10.1109/CVPR.2016.91
14. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Fei-Fei, L. (2015). ImageNet large scale visual recognition challenge. International Journal of Computer Vision, 115(3), 211-252. https://doi.org/10.1007/s11263-015-0816-y

**Appendix 1- Screenshots and steps**

**Link to the dataset:** [**https://www.kaggle.com/datasets/tongpython/cat-and-dog**](https://www.kaggle.com/datasets/tongpython/cat-and-dog)

**Fig 1.1 Basic Architecture of CNN (Phung, & Rhee, 2019).**

A diagram of a process

Description automatically generated

**Fig 1.2 Process of Building CNN - (Roy, A. 2020, July 1)**

A diagram of a number of squares

Description automatically generated

“Pls, note: Although, I have added all the codes on Appendix 2 section, but below is just a screenshot of snippet of my codes that can help and guide anyone in replication of my”

**Fig 1.3: Environment setup - Step 1: Install & Import Necessary Libraries**

**A screenshot of a computer

Description automatically generated**

**Fig 1.3b Install Libraries**

**A screenshot of a computer program

Description automatically generated**

**Fig 1.4 Step 2: Define Directories for Dataset**

**A screen shot of a computer

Description automatically generated**

**Fig 1.5 Step 3: Count Images in Folders**

**A screen shot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated**

**Fig 1.6 Step 5: Visualize Data Distribution**

**A computer code on a black background

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Fig 1.7 Step 6: Data Preparation**

**A screenshot of a computer program

Description automatically generated**

**A black screen with a black border

Description automatically generated**

**Fig 1.8 Step 7: Visualize Preprocessed Images**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer screen

Description automatically generated**

**Fig 1.8 - Step 8: Build and Train a Simple CNN**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Fig 1.9 Step 9: Confusion Matrix for Simple CNN**

**A screenshot of a computer

Description automatically generated**

**Fig 1. 10 - Step 11: Download and Fine-Tune VGG16**

**A screenshot of a computer

Description automatically generated**

**Fig 1.11 Step 11 - Post preprocess visualization with vgg16**

**A screenshot of a computer

Description automatically generated**

**Fig 1.12 Valid range for imshow with RGB**

**A screenshot of a computer

Description automatically generated**

**Fig 1.13 Confusion matrix, without normalization**

**A screenshot of a computer

Description automatically generated**

**Table 1.1 Result table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Epoch** | **Accuracy** | **Loss** | **Val Accuracy** | **Val Loss** |
| 1 | 0.6297 | 3.7569 | 0.639 | 0.7978 |
| 2 | 0.8531 | 0.3398 | 0.669 | 0.8285 |
| 3 | 0.9626 | 0.1133 | 0.71 | 0.827 |
| 4 | 0.9914 | 0.0428 | 0.708 | 0.9441 |
| 5 | 0.9968 | 0.0205 | 0.694 | 1.1308 |
| 6 | 0.9791 | 0.0648 | 0.678 | 1.1853 |
| 7 | 0.9679 | 0.0901 | 0.68 | 1.1358 |
| 8 | 0.9851 | 0.0505 | 0.681 | 1.5538 |
| 9 | 0.9844 | 0.0476 | 0.704 | 1.3721 |
| 10 | 0.9894 | 0.0306 | 0.689 | 1.3557 |

**Appendix 2- Code (Click on the code below to view the entire code)**

