

A black and white rectangle

Description automatically generatedA black and white rectangle

Description automatically generated

**Using RGB-Encoded Audio Features for Convolutional Models**

**Speech Emotion Recognition (S.E.R)**

***KV6003 – Individual Computing Project***

***Panagiotis Theodorou – W20020633***

***Supervisor: Dr Hua Mao***

***2nd Marker: Dr Nic Whitton***

***Word Count: XXXX***

***Date: DD/MM/YYYY***

**2023 - 2024**

**DECLARATION OF AUTHORSHIP**

**DECLARATIONS**

I declare the following:

(1) that the material contained in this dissertation is the end result of my own work and that due acknowledgement has been given in the bibliography and references to **ALL** sources be they printed, electronic or personal.

(2) the Word Count of this Dissertation is ...................................

(3) that unless this dissertation has been confirmed as confidential, I agree to an entire electronic copy or sections of the dissertation to being placed on the eLearning Portal (Blackboard), if deemed appropriate, to allow future students the opportunity to see examples of past dissertations. I understand that if displayed on eLearning Portal it would be made available for no longer than five years and that students would be able to print off copies or download.

(4) I agree to my dissertation being submitted to a plagiarism detection service, where it will be stored in a database and compared against work submitted from this or any other School or from other institutions using the service.

In the event of the service detecting a high degree of similarity between content within the service this will be reported back to my supervisor and 2nd marker, who may decide to undertake further investigation that may ultimately lead to disciplinary actions, should instances of plagiarism be detected.

(5) I have read the Northumbria University/Engineering and Environment Policy Statement on Ethics in Research and Consultancy and I confirm that ethical issues have been considered, evaluated and appropriately addressed in this research.

SIGNED:

**Acknowledgments:**

* Acknowledge any individuals, organizations, or resources that contributed to your research.

**Abstract:**

This paper investigates the utilisation of RGB-encoded audio features for speech emotion recognition, a unique approach aimed at enhancing the capabilities of artificial intelligence in interpreting human emotions by investigating a multitude of models, by incorporating auditory data alone. Traditional audio signal processing methods for emotion recognition often struggle with capturing the subtleties of emotional expressions effectively. By transforming audio features such as Mel-Frequency Cepstral Coefficients (MFCC), Mel-Spectrograms, and Chroma into RGB images, this study leverages the advanced pattern and texture recognition capabilities of convolutional neural networks (CNNs) tailored for image analysis. The research demonstrates that converting audio data into a visual format allows for improved utilisation of CNN architectures, such as VGG, ResNet, and DenseNet, which are inherently more adept at handling visual inputs. This transformation leads to enhanced model performance, showcasing a significant increase in accuracy when classifying various emotional states across diverse datasets. Additionally, RGB encoding simplifies the data processing pipeline by reducing data complexity and computational demands, thereby increasing efficiency and scalability. However, the potential drawbacks, including the risk of losing critical auditory information and the varying effectiveness across different emotional and linguistic contexts, are also discussed. The paper concludes with recommendations for future work, emphasizing the need to integrate multimodal data sources and explore higher-dimensional data representations to further refine and expand the utility of RGB-encoded audio features in emotion recognition tasks. This study sets a foundational framework for future investigations, suggesting that the strategic use of RGB-encoded audio data can lead to more integrated, efficient, and accurate emotion recognition systems, ultimately enhancing the interaction between humans and AI systems.

Table of Contents

[1 Introduction 1](#_Toc165840387)

[1.1 Overview 1](#_Toc165840388)

[1.2 Objectives 2](#_Toc165840390)

[2 Literature Review 3](#_Toc165840391)

[2.1 Studies of Models in Speech Emotion Recognition 3](#_Toc165840392)

[2.1.1 Deep Learning Models 3](#_Toc165840393)

[2.1.2 Machine Learning Models 3](#_Toc165840394)

[2.1.3 Pre-Trained Models 4](#_Toc165840395)

[2.2 Challenges and Limitations 5](#_Toc165840401)

[2.2.1 Datasets 5](#_Toc165840402)

[2.2.1.1 CREMA-D 5](#_Toc165840403)

[2.2.1.2 RAVDESS 6](#_Toc165840404)

[2.2.1.3 SAVEE 6](#_Toc165840405)

[2.2.1.4 EMO-DB 6](#_Toc165840406)

[2.2.2 Emotions 7](#_Toc165840407)

[2.2.3 Authenticity 7](#_Toc165840410)

[2.2.4 Cultural and Biological Biases 7](#_Toc165840413)

[2.2.5 Overfitting 8](#_Toc165840416)

[3 Experimentation 9](#_Toc165840419)

[3.1 Test Plan 9](#_Toc165840420)

[3.2 Performance 10](#_Toc165840447)

[3.2.1 Vector Features 10](#_Toc165840448)

[3.2.2 Image Features 12](#_Toc165840456)

[3.3 Selection 14](#_Toc165840472)

[3.3.1 Model Selection Justification: CNN Family Models 14](#_Toc165840473)

[3.3.2 Dataset Selection Rationale: EMODB & RAVDESS 14](#_Toc165840474)

[3.3.3 Optimisation 15](#_Toc165840475)

[4 Methodology 16](#_Toc165840476)

[4.1 Data Split and Structure 16](#_Toc165840477)

[4.2 Audio Feature Extraction and Conversion 16](#_Toc165840481)

[4.2.1 Feature Extraction 17](#_Toc165840485)

[4.2.2 RGB Conversion 17](#_Toc165840490)

[4.3 Re-Imagined Datasets 18](#_Toc165840496)

[4.4 Augmentation 19](#_Toc165840540)

[4.4.1 Second-Stage Feature Extraction 20](#_Toc165840543)

[4.4.2 Clustering 20](#_Toc165840546)

[4.5 Classification Models 20](#_Toc165840549)

[4.5.1 DenseNet121 20](#_Toc165840551)

[4.5.2 EfficientNetB0 21](#_Toc165840552)

[4.5.3 ResNet50 22](#_Toc165840553)

[4.5.4 VGG16 22](#_Toc165840554)

[4.6 Optimisation 22](#_Toc165840555)

[4.6.1 Hyperparameters 23](#_Toc165840557)

[4.6.2 Layers & Weights 23](#_Toc165840561)

[4.6.3 Genetic Algorithm 24](#_Toc165840563)

[5 Results 24](#_Toc165840566)

[5.1 Frozen Layers 24](#_Toc165840567)

[5.2 Unfrozen Layers 26](#_Toc165840585)

[5.3 Per-Variant 28](#_Toc165840605)

[6 Discussion 31](#_Toc165840626)

[6.1 Interpretation and Challenges 31](#_Toc165840627)

[6.2 Confusion of Emotional Categories and Future Directions 31](#_Toc165840628)

[6.3 Performance Insights and Implications 31](#_Toc165840629)

[7 Conclusion 31](#_Toc165840630)

[7.1 Advantages 32](#_Toc165840631)

[7.2 Potential Weaknesses 32](#_Toc165840632)

[7.3 Future Work 33](#_Toc165840633)

[7.4 Final Thoughts 33](#_Toc165840634)

[**References:** 34](#_Toc165840635)

[**Appendices:** 35](#_Toc165840636)

# Introduction

In the ever-evolving landscape of artificial intelligence, the interpretation of human emotions through speech, facilitated by sophisticated SER systems, stands as a cornerstone of technological advancement. Among the most innovative techniques in this field is the use of RGB-Encoded audio feature images. This method, which transforms auditory information into a visual spectrum, not only enhances the interpretability of data but also significantly broadens the application horizon of SER systems in fields as diverse as healthcare, customer service, and security (Kerkeni et al., 2021). Historically, speech emotion recognition has transitioned from rudimentary analog systems to complex AI-driven technologies, marking a significant evolution in the field (Schuller et al., 2011). This transformation reflects a broader shift from traditional signal processing to sophisticated learning algorithms that capitalize on the nuances of human speech, offering a more refined understanding of underlying emotional states.

This report explores the integration of RGB-Encoded audio features within SER systems. It is structured to first introduce the basic concepts of SER with its multitude of potential models that contribute to SER systems, followed by an in-depth examination of Deep Learning and Machine Learning models, among some basic performance metrics, and pursue with the methodology and analysis of the RGB-Encoded audio features and their comparative effectiveness against traditional methods. The paper will conclude with a detailed analysis and discussion of the AI models and their performance metrics, and the effectiveness that the RGB-Encoded audio features have played, highlighting their applicability based on future advancements and studies.

## Overview

## Speech Emotion Recognition (SER) serves as a pivotal technology within human-computer interaction, enabling machines to infer human emotions from vocal cues. This domain synthesizes knowledge from linguistics, psychology, and computational technology to parse nuances in speech such as tone, pitch, and rate, ultimately deducing the speaker’s emotional state. Emotions, as defined by psychological research, are complex states elicited by perceived significant personal experiences, which are reflected subtly in speech patterns (Williams & Happé, 2010).

The advancement of Speech Emotion Recognition (SER) technologies has been closely tied to innovations in artificial intelligence, particularly through the integration of sophisticated machine learning algorithms such as neural networks. Kerkeni et al. (2021) detail how these algorithms have been pivotal in enhancing SER systems by learning and interpreting complex patterns in speech data. Their research particularly highlights the role of Mel-frequency cepstral coefficients (MFCCs) and modulation spectral features in capturing the emotional content embedded within voice signals. MFCCs, which provide a reliable representation of the power spectrum of sound, mimic the human auditory system's response and are instrumental in recognizing and processing speech. Modulation spectral features, on the other hand, offer insights into the variations in speech modulation patterns, which are crucial indicators of emotional states.

The study by Kerkeni et al. (2021) demonstrates the effectiveness of these features combined with recurrent neural network (RNN) classifiers to improve the accuracy of emotion recognition systems. They explored the use of different classifiers, including multivariate linear regression and support vector machines, alongside RNNs, showing substantial improvements in recognition accuracy when using speaker normalization and feature selection techniques. This combination of neural networks for pattern recognition and detailed feature extraction from speech signals significantly propels the field of SER forward. By improving the functional accuracy of SER systems, this integration extends their applicability across diverse domains where understanding and interacting with human emotions are crucial, such as in customer service, therapy, and educational technology.

The implications of SER are profound across multiple sectors. In healthcare, SER can detect nuances in patient communications, potentially identifying distress or pain. It transforms customer service by analysing caller emotions to tailor agent responses, thereby enhancing customer satisfaction. Furthermore, in educational settings, SER can monitor and adapt to the emotional state of learners, potentially improving engagement and educational outcomes (Schuller et al., 2011), in which supports the notion that SER technologies are not just technical achievements but have significant practical applications that can transform various industries.

## Objectives

The objectives of this research are centred around enhancing the effectiveness of SER systems through innovative data interpretation and model comparisons. This research intends to delve into the integration and performance evaluation of various AI models within SER, particularly focusing on RGB-Encoded audio features. These features represent a novel approach in transforming audio signals into a visual format that can be more effectively processed by neural networks, providing a fresh perspective on emotion recognition, by essentially working with 3 features for the price of 1 image, for each file.

The scope of the study is explicitly defined to explore and validate the performance of these models across different datasets, emphasizing the potential of audio-only data, with the unique approach of data interpretation. This focus addresses the challenge of extracting reliable emotional cues from speech without relying on supplementary modalities like facial expressions, physiological signals, or metadata, which are often used in multimodal emotion recognition systems. Anticipated contributions of this research include demonstrating the effectiveness and particular niches where RGB-Encoded audio features might outperform traditional features. Moreover, by comparing various deep learning and machine learning models, this study aims to identify optimal approaches that balance accuracy with computational efficiency, suitable for real-time applications.

# Literature Review

## Studies of Models in Speech Emotion Recognition

### Deep Learning Models

Deep Learning models, as explored in the article by Fayek, Lech, and Cavedon (2017), demonstrate significant capabilities in Speech Emotion Recognition (SER). These models can effectively handle both static and dynamic classification problems, making them versatile for various applications. The employment of minimal speech processing and end-to-end deep learning architectures allows for the modelling of intra-utterance dynamics, which is crucial for recognising emotional states from speech. The benefits of using Deep Learning for SER include achieving state-of-the-art results on the IEMOCAP database for speaker-independent SER, demonstrating the models' ability to generalise across different speakers. Furthermore, these models present a simple pipeline and low latency in emotion recognition tasks, which are advantageous for real-time applications. The exploration of feed-forward and recurrent neural network architectures, along with their variants, illuminates their strengths and limitations in handling paralinguistic elements of speech, offering insights into the future development of more sophisticated and efficient SER systems (Fayek, H.M., Lech, M. & Cavedon, L., 2017).

Pandey et al. (2019) explored the use of deep learning techniques in recognizing emotions from speech, highlighting the significance of enabling machines not only to understand the content but also to interpret the emotional nuances of human speech. Unlike traditional machine learning models, which are limited in their ability to capture the complexities of emotional states, deep learning offers a more effective approach by automatically learning high-level features from data.

In their evaluation, Pandey et al. (2019) used "Weighted Accuracy" to compare different deep learning architectures and input features across four basic emotions: Neutral, Angry, Happy, and Sad. Their findings indicated that the best results were obtained using CNN, LSTM, and a hybrid model combining CNN and BLSTM. To ensure the robustness of their findings and mitigate any potential bias from a single dataset, they utilised a five-fold cross-validation approach with two datasets, Emo-DB and IEMOCAP. The hybrid model (CNN+BLSTM) with Mel-Frequency Cepstral Coefficients (MFCC) as the input feature achieved the highest accuracy (82.35%) on the Emo-DB dataset. This model, however, faced challenges in accurately classifying happy emotions, attributed to class imbalances and the arousal similarities between happy and angry utterances. Conversely, for the IEMOCAP dataset, the same hybrid model with a Mel-Spectrogram input feature performed best. Interestingly, it accurately classified "happy" emotions more effectively than others, which Pandey et al. (2019) suggest may be due to the more natural elicitation of emotions in the IEMOCAP dataset, despite an overall accuracy below 50%.

|  |
| --- |
|  |
| ***Fig 1:*** *Confusion Matrices of Pandey et al. (2019) evaluations of both datasets across 4 emotions in a five-fold validation.* |

### Machine Learning Models

Machine learning models play a pivotal role in the development of Speech Emotion Recognition (SER), which aims to decipher human emotions from spoken language. Traditional machine learning models, distinct from their deep learning counterparts, bring to SER a mix of capabilities and challenges. The efficacy of these models (Support Vector Machines (SVM), Hidden Markov Models (HMM), K-nearest neighbours (KNN) and more) in interpreting emotional nuances in speech is underscored by their specific strengths and limitations. These models' contributions to SER, juxtaposed against their operational drawbacks, illustrate the nuanced landscape of machine learning applications in understanding human affective states through speech.

Support Vector Machines (SVM) are renowned for their effectiveness in classification tasks, including SER. SVMs operate by finding the optimal hyperplane that separates different emotion classes in a feature space. Lin and Wei (2005) utilised SVM in their SER system, demonstrating its capability to classify emotional states with high accuracy (Lin and Wei, 2005). However, the model is primarily binary and struggles with multi-class emotion datasets. It also suffers from long processing times and decreased accuracy in the presence of background noise (Anusha et al., 2021). Despite these drawbacks, the model's high accuracy in binary classifications positions it as a valuable tool in SER where binary emotion classification is sufficient.

Hidden Markov Models (HMM) offer another approach by modelling the temporal sequence of speech features. HMMs are particularly adept at capturing the dynamic nature of speech, making them suitable for SER applications. Aouani and Ayed (2020) demonstrated the use of HMM in SER, achieving impressive recognition rates by capturing the temporal dynamics of speech features (Aouani and Ayed, 2020). However, HMMs face challenges in feature selection, as the selected features may not fully represent the emotional state conveyed in speech. Moreover, the complexity of HMMs can lead to increased computational requirements.

K-nearest neighbours (KNN) is a simpler model that has been applied to SER. Its main advantage lies in its simplicity and interpretability. KNN classifies emotions based on the closest training examples in the feature space. While this model is easy to implement and understand, its accuracy is generally lower compared to more complex models like SVM and HMM. Additionally, KNN's performance is highly dependent on the choice of k and the distance metric used, which may require fine-tuning for optimal performance in SER tasks (Anusha et al., 2021).

Each of these machine learning models brings unique strengths to SER. SVMs are highly effective in binary classifications and can achieve high accuracy levels. HMMs excel in capturing the temporal dynamics of speech, making them suitable for analysing the time-dependent aspects of emotions in speech. Meanwhile, KNN offers simplicity and ease of interpretation, although it may not always achieve the highest accuracy.

However, these models also face limitations. The binary nature of SVMs limits their application in multi-class emotion recognition tasks, and their performance can be significantly affected by noise. HMMs, while powerful, can be complex and computationally demanding. KNN's performance is variable and often inferior to more sophisticated models.

### Pre-Trained Models

Pre-trained models have become a cornerstone in advancing speech emotion recognition (SER) systems, primarily due to their ability to generalize well from extensive pre-existing data, thus enhancing system robustness, efficiency, and adaptability across various settings.

DenseNet, a model renowned for its efficient feature reuse, has shown particular promise in SER applications. Latif et al. (2020) highlights its superiority over other architectures like ResNet, attributing this to DenseNet's unique ability to enhance feature propagation throughout the network. This characteristic is crucial in maintaining high performance even under adverse conditions such as noisy environments or during adversarial attacks, where traditional models might falter. By integrating DenseNet with LSTM and Highway networks, the study not only leverages temporal features crucial for understanding emotional nuances in speech sequences but also enhances the robustness of the SER system against external disruptions. This integration underscores the model's capability in handling dynamic and temporally complex audio data (Latif, S., Rana, R., Khalifa, S., Jurdak, R., and Schuller, B.W., 2020).

Expanding the utility of pre-trained models, Ottl et al. (2020) utilise VGG alongside DenseNet to extract deep spectrum features from audio signals. This approach adapts methodologies typically reserved for image data, applying them to spectrograms of audio samples to effectively capture the textural nuances that are indicative of different emotions. The success of these models in decoding complex patterns from spectrogram images illustrates the adaptability of convolutional networks, originally designed for visual data, in parsing and learning from auditory information. The effectiveness of these models in a group-level SER context further demonstrates their robustness and the potential to enhance collective emotional intelligence applications (Ottl, S., Amiriparian, S., Gerczuk, M., Karas, V., and Schuller, B., 2020).

Chakhtouna et al. (2022) explore a broader spectrum of pre-trained models, including ResNet, VGG, and EfficientNet, emphasizing the versatility of transfer learning in SER. The study shows how these models, initially designed for image classification tasks, can be fine-tuned to recognize emotional states from speech. Each model brings unique strengths to SER—EfficientNet, for example, is lauded for its scalability and lower computational demand, making it suitable for mobile applications where resources are limited. The fine-tuning process allows these models to adapt to the intricacies of emotional speech, tailoring their originally broad capabilities to meet the specific demands of emotion recognition. This adaptability is key to developing effective SER systems that are both accurate and efficient in real-world applications (Chakhtouna, A., Sekkate, S., and Adib, A., 2022).

Through the strategic use of pre-trained models, SER systems can achieve remarkable improvements in accuracy and efficiency. These models not only bring the benefit of advanced feature learning capabilities but also introduce methodological innovations that expand the horizons of what SER systems can achieve. By leveraging the complex, pre-learned feature representations these models offer, researchers and developers can significantly enhance the robustness and reliability of emotion recognition systems.

## Challenges and Limitations

### Datasets

Developing a Speech Emotion Recognition (SER) model in Artificial Intelligence presents challenges, notably in the quality and scope of the datasets used. High-quality audio is essential for accurate emotion classification, as poor audio quality, background noise, or inaccurate labels can significantly degrade model performance. The quantity of data is also critical; robust models require extensive datasets to accommodate the diversity of speech patterns. Yet, acquiring such large datasets is resource-intensive and often impractical, leading to potential overfitting with smaller datasets.

Metadata about speakers, recording conditions, and emotional states is vital for contextualizing data, but the absence of detailed metadata challenges models to learn nuanced speech differences. This study aims to surmount this by generalizing emotion recognition without relying on metadata, focusing on more generalised feature extraction challenges. Identifying relevant speech features, like the timbral aspects ,pitch, and harmonic nuances, is crucial for effective emotion classification. Datasets frequently lack diversity, limiting a model's ability to generalize across various demographics and real-world situations, presenting another layer of complexity in SER development.

#### CREMA-D

The CREMA-D (Crowd-sourced Emotional Multimodal Actors Dataset) dataset, established by Cao et al. (2014), serves as a comprehensive tool for the study of speech emotion recognition. It includes 7,442 video clips from 91 diverse actors—48 men and 43 women aged 20 to 74 years, representing various ethnic backgrounds. These clips capture the actors expressing six basic emotions: happiness, sadness, anger, fear, disgust, and neutrality, with each emotion displayed at different intensity levels, providing a broad spectrum of emotional states for analysis. Each video clip contains spoken dialogue in a neutral setting to highlight emotional expressions without contextual bias, focusing on vocal characteristics like tone and pitch. This controlled environment is vital for accurately identifying emotions based solely on vocal cues.

A unique feature of CREMA-D is its emotion labelling process, which utilises ratings from a large crowd-sourced panel of 2,443 individuals, rather than a limited group of experts. This method enhances the dataset's reliability and relevance across various demographics and cultural contexts. For audio-based emotion recognition research, the dataset's audio files are particularly valuable. They include a diverse range of vocal expressions, making CREMA-D an essential resource for training and evaluating machine learning and deep learning models in speech emotion recognition.

#### RAVDESS

The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS), created by Livingstone and Russo (2018), is a key resource in speech emotion recognition, notable for its dual modality which includes both audio and visual emotional expressions. This feature allows for the examination of multimodal influences on emotion recognition systems. RAVDESS is composed of 7,356 recordings from 24 professional actors (12 male and 12 female), who depict a range of emotions—calm, happy, sad, angry, fearful, surprise, and disgust at two intensity levels, plus a neutral baseline. This variety enriches the dataset with a broad spectrum of emotional states, aiding in the development of sophisticated emotion recognition models. Each recording undergoes a detailed assessment by North American evaluators who rate the emotional validity, intensity, and genuineness, enhancing the dataset’s reliability and making it a valuable benchmark for training machine learning models in emotion recognition.

#### SAVEE

The Surrey Audio-Visual Expressed Emotion (SAVEE) Database, established by Jackson and Haq (2014), is pivotal in the field of speech emotion recognition. It provides audio-visual recordings of four male actors delivering British English utterances across seven emotions: anger, disgust, fear, happiness, sadness, surprise, and neutral, totalling 480 utterances. These were carefully selected from the TIMIT corpus to maintain a phonetic balance, essential for effective emotion recognition systems.

Recorded in a visual media lab with advanced audio-visual technologies, SAVEE emphasizes high-quality, phonetically balanced captures of diverse emotional expressions. The dataset’s reliability and effectiveness are underscored by evaluations from ten subjects under audio, visual, and audio-visual conditions, with recognition rates of 61%, 65%, and 84% respectively, demonstrating its robust potential for developing sophisticated emotion recognition systems.

SAVEE features recordings from four native English-speaking male actors, aged 27 to 31, ensuring a variety of vocal characteristics. It encompasses a comprehensive emotional range in 480 utterances, aiming for precise emotion recognition and analysis. This dataset is invaluable for researchers focused on both unimodal and multimodal emotion recognition approaches, significantly advancing AI’s ability to understand human emotions through speech and enhancing human-computer interaction.

#### EMO-DB

The Emo-DB dataset, created by Burkhardt et al. (2005) during a DFG-funded project between 1997 and 1999, offers a valuable collection of emotional utterances spoken by actors, recorded in the anechoic chamber of the Technical University Berlin. This dataset encompasses over 500 utterances, distinctly categorized by emotions, with each utterance accessible for research through a detailed web interface. This interface not only allows users to filter utterances by speaker, text, and emotion but also provides essential data such as syllable labels, duration, intonation contours, and results from various perception tests. Emo-DB is structured to facilitate a thorough analysis of emotional speech, with tools to measure fundamental frequency, energy, loudness, duration, stress, and rhythm. It also includes label files that cover syllable and phone labels, and perception test results related to emotion recognition, naturalness, syllable stress, and emotional intensity. This rich dataset aids researchers in comprehensively understanding how emotional expressions are perceived and processed.

### Emotions

The complexity and variability of human emotions pose significant challenges in Speech Emotion Recognition (SER). Emotions are inherently subjective experiences that can vary widely among individuals, even in similar situations. This variability makes it difficult to label emotions accurately in speech datasets, as what is perceived as anger by one individual may be interpreted as frustration by another. Additionally, emotions can overlap; for example, excitement and anger can share similar physiological expressions such as an elevated tone of voice or rapid speech, leading to confusion in classification tasks.

The subtlety with which emotions are expressed also adds layers of complexity. For instance, the same emotion can be expressed with different intensities, and subtle emotional expressions are often harder to detect and categorize accurately. Furthermore, emotions can transition rapidly during speech, making it challenging for static models to track these changes effectively. These factors contribute to the intricate task of developing SER systems that can accurately recognize and differentiate between a broad spectrum of emotional states, often leading to the need for highly sophisticated modelling techniques that can handle such complexities.

### Authenticity

Creating an authentic dataset for SER is fraught with challenges. Most datasets are constructed by asking actors to simulate emotions, which can sometimes result in exaggerated or unnatural expressions that do not truly represent genuine emotional states. This lack of authenticity can lead to models that perform well on staged data but fail to generalize to real-world scenarios where emotional expressions are more nuanced and less pronounced.

Legal and ethical considerations further complicate the creation of SER datasets. Recording genuine emotional reactions can involve scenarios that might be psychologically distressing, posing ethical concerns regarding the mental well-being of participants. Additionally, the need to obtain informed consent and ensure privacy can limit the scope of emotions that can be ethically elicited and recorded, thus constraining the diversity and richness of the dataset. These issues not only impact the quality of data but also raise significant moral questions about the methods used to collect emotional speech data.

### Cultural and Biological Biases

Cultural and linguistic diversity presents another layer of complexity in SER. Different cultures and languages may express or interpret emotions in varied ways, which can lead to misinterpretations when SER systems are applied across different demographic groups. For instance, the tonal nuances of a language like Mandarin can lead to different emotional interpretations compared to a non-tonal language like English. Similarly, cultural norms influence how openly and intensely emotions are expressed, which can affect the accuracy of emotion recognition systems that are not tuned to these nuances.

Gender biases also play a role, as societal norms about emotional expression can vary significantly between genders within the same culture. Biological differences, such as variations in voice pitch and intensity between genders, can further skew the performance of SER systems if not properly accounted for. These biases must be addressed to develop SER systems that are fair and effective across different populations.

### Overfitting

Overfitting is a prevalent challenge in SER, particularly because of the complex nature of audio data processing. Audio signals contain a wealth of information that is often irrelevant to the task of emotion recognition, such as background noise or variations in speech pace. Extracting meaningful features from raw audio data requires sophisticated preprocessing techniques and feature engineering to ensure that the model focuses on relevant signals indicative of emotional content.

Moreover, the availability of large and diverse audio datasets is limited, which can lead models to overfit to the peculiarities of the data they are trained on, rather than learning to generalize across different sets of speech data. This limitation can severely hinder the model's performance in real-world applications, where the variability of input data is significantly higher than that of the training set. Sophisticated techniques such as data augmentation, regularization, and transfer learning are essential to mitigate overfitting and enhance the robustness and generalizability of SER systems.

# Experimentation

## Test Plan

The experimentation phase of this project involves a comprehensive test plan designed to assess the effectiveness of both traditional machine learning and deep learning models on speech emotion recognition (SER). Here’s the step-by-step approach:

* **Model Selection**
  + Identify a range of traditional machine learning models such as Support Vector Machines (SVM), Decision Trees, and k-Nearest Neighbours (k-NN), as well as deep learning models including Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN with LSTM), and ResNet. The choice of models will be dictated by their historical performance on audio processing tasks.
* **Feature Extraction**
  + Mel-Spectrograms: Utilise Mel-Spectrograms which are a visual representation of the spectrum of frequencies in a sound as they vary with time.
  + Mel Frequency Cepstral Coefficients (MFCCs): Employ MFCCs which summarise the frequency distribution across the window size, ideal for capturing the time-domain aspects of the audio signal for traditional machine learning models.
* **Dataset Utilization:** Conduct experiments using four distinct audio datasets:
  + CREMA-D
  + EMO-DB
  + RAVDESS
  + SAVEE

Each dataset will be tested separately to isolate dataset-specific performance and identify any biases or limitations inherent in the datasets.

* **Testing Strategy**
  + Vector Input Features: Test each model using vector input features (both machine learning and deep learning models) derived from Mel-Spectrograms and MFCCs to assess how well traditional feature extraction techniques perform across different model architectures.
  + Image Input Features: Apply image input features (only for deep learning models) using both Mel-Spectrograms, and MFCCs.
* **Evaluation and Reporting**
  + Generate classification reports for each experiment to evaluate the models on metrics such as accuracy, precision, recall, and F1-score. This step is crucial for determining which models exhibit the potential for further development.
  + Compare the performance of models across different datasets to select two datasets that show the most promise for advancing to more sophisticated SER models.
  + Dataset and Feature Augmentation:
  + Develop an RGB-Encoded dataset where audio features are encoded as RGB image data, enhancing the dataset with more sophisticated feature augmentation processes to better capture the nuances of emotional expressions in speech.
* **Model Optimization**
  + Optimise the selected models using techniques such as hyperparameter tuning, network architecture adjustments, and possibly ensemble methods to enhance their performance and robustness.
* **Advanced Analysis**
  + Analyse the results of the optimised models under two scenarios:
  + ALL Class Classification: Where all emotional categories are considered.
  + 4-Class Classification: Simplifying the problem to four major emotional states (happy, sad, angry, neutral) to see if model performance improves in a less complex scenario.

This test plan is structured to methodically evaluate the capabilities of various SER models using both traditional and innovative feature extraction methods, across multiple datasets. The final goal is to refine and develop a more sophisticated SER system that is robust, accurate, and generalizable across different emotional states and datasets.

## Performance

### Vector Features

|  |  |
| --- | --- |
| A graph with numbers and lines  Description automatically generated with medium confidence | A black screen with colorful lines  Description automatically generated |
| *Fig 2: Comparisons of all 4 datasets F1 Macro-Average Performance Metrics, in Deep Learning and Machine Learning models, using Mel-Spectrogram and MFCC as a input Vector Features* | |

In assessing the effectiveness of various artificial intelligence models for Speech Emotion Recognition (SER), a detailed analysis was conducted using vector input features of Mel-Spectrogram and MFCC across four datasets: CREMA-D, EMO-DB, RAVDESS, and SAVEE. The F1-Score "Macro Average" served as a primary metric for this evaluation, given its relevance in reflecting model performance across diverse emotional expressions within these datasets.

The findings indicate a general trend where models tend to exhibit superior performance on the EMO-DB and SAVEE datasets when utilizing Mel-Spectrogram features. Artificial Neural Networks (ANN), Multi-Layer Perceptron (MLP), and Support Vector Machines (SVM) demonstrated enhanced accuracy across these datasets. This trend suggests that Mel-Spectrogram features, which offer a detailed representation of the spectral texture of sounds, align well with the models' capabilities in discerning nuanced emotional states from speech.

A noticeable performance decline was observed for most models when employing MFCC features, especially pronounced in the CREMA-D and EMO-DB datasets. This reduction could be attributed to the inherent differences in the feature representations, where Mel-Spectrograms provide a granular view that may capture emotional nuances more effectively compared to the potentially more abstracted or compressed information conveyed by MFCCs.

|  |  |
| --- | --- |
| A graph of different colored bars  Description automatically generated | A graph of different colored bars  Description automatically generated |
| *Fig 3:* F1-Score Macro-Average Performance Comparisons of Each Deep Learning and Machine Learning Model, from each Dataset, relative to each Feature Vector (MFCC vs Mel-Spectrogram) | |

Among the models evaluated, Hidden Markov Models (HMM) and Gaussian Mixture Models (GMM) consistently underperformed across both feature types and all datasets. This outcome could highlight limitations in these models' ability to leverage the dynamic, temporal aspects of speech essential for accurate emotion recognition, underlining the importance of model selection in conjunction with appropriate feature utilization for SER tasks.

Delving deeper into the Mel-Spectrogram vector metrics with an emphasis on the F1 Macro Average reveals distinct patterns. For instance, in the CREMA-D dataset, MLP distinguished itself with the highest F1 Macro Average, suggesting its effectiveness in leveraging the intricate details captured by Mel-Spectrogram features. On the contrary, HMM displayed significantly lower performance, underscoring its struggles with processing Mel-Spectrogram features for emotion recognition tasks.

Similar observations were made across the other datasets, with MLP consistently emerging as a top performer, showcasing its robustness in handling Mel-Spectrogram features effectively. ANN and SVM also showed commendable performance, positioning themselves as viable alternatives for SER applications requiring high F1 Macro Averages.

Transitioning to the analysis of MFCC vector metrics, while a general performance dip was noted compared to Mel-Spectrograms, ANN managed to maintain a leading position in several instances. This resilience underscores ANN's adaptability to different auditory feature representations, a critical capability for SER systems.

Through this comparative analysis, the crucial role of feature selection in SER becomes evident, with Mel-Spectrogram features proving particularly advantageous for enhancing model performance. Furthermore, the reliance on the F1 Macro Average as a key metric offers a nuanced perspective on model efficacy, particularly valuable in the context of imbalanced datasets. The standout performance of MLP with Mel-Spectrograms and the overall resilience of ANN across different feature types highlight their potential utility in the ongoing development of SER technologies. This analysis not only informs the selection of models and features for SER but also lays the groundwork for future advancements in the field.

### Image Features

|  |  |  |
| --- | --- | --- |
|  | Mel-Spectrograms | MFCCs |
| CREMA-D |  |  |
|  |  |  |
| EMO-DB |  |  |
|  |  |  |
| RAVDESS |  |  |
|  |  |  |
| SAVEE |  |  |
| Fig 4: Comparison charts of Mel-Spectrogram and MFCC images, independently, using F1-Score Macro Average metric on all resolutions, for each dataset across all 5 deep learning models. | | |

Evaluating model performance in Speech Emotion Recognition (SER) through image input features, particularly Mel-Spectrograms and MFCCs at varied resolutions (32x32, 64x64, 128x128, 256x256), provides key insights into the influence of image resolution and feature type across four datasets: CREMA-D, EMODB, RAVDESS, and SAVEE. The study focuses on the application of deep learning models to discern trends that could inform future SER methodologies.

A notable trend is the correlation between increased image resolution and enhanced model performance. Higher resolutions capture more detailed information, crucial for the accurate recognition of emotions from audio data. This trend underscores the importance of detailed visual representations in distinguishing complex emotional states, with Mel-Spectrogram features consistently outperforming MFCC features across most datasets and resolutions. Mel-Spectrograms, with their rich time-frequency details, offer a comprehensive view of the audio data's emotional content, suggesting their superiority over the potentially more abstract MFCC features in SER tasks.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **32x32** |  |  | **64x64** |  |
|  |  |  |  |  |
| **128x128** |  |  | **256x256** |  |
| ***Fig 5:*** *Comparison charts of Mel-Spectrogram and MFCC images,, using F1-Score Macro Average metric on all resolutions independently, across all models, with all datasets together.* | | | | |

The analysis across resolutions reveals:

* At **32x32**, CNN models using Mel-Spectrograms particularly on the EMODB dataset, exhibit superior F1 Macro Averages. This resolution also highlights the challenges faced by models using MFCC features on the SAVEE dataset, indicating the limitations of lower-resolution images and MFCC features in capturing emotional nuances.
* The shift to **64x64** resolution marks a significant improvement in model performance across all datasets, with CNN models using Mel-Spectrograms on the RAVDESS dataset showing noteworthy enhancements. This resolution begins to unveil the potential utility of MFCC features, with improved performance in models like CNN on the RAVDESS dataset.
* At **128x128**, the performance jump is particularly evident on the EMODB dataset with CNN models using Mel-Spectrograms. DenseNet models employing MFCC features on this dataset also experience substantial improvements, showcasing the advantages of increased resolution for both feature sets.
* The highest performances are observed at the **256x256** resolution, with models such as ResNet on the CREMA-D dataset using Mel-Spectrograms and DenseNet on the EMODB dataset for both features achieving remarkable F1 Macro Averages. However, the performance of MFCC features at this resolution is mixed, with some models experiencing performance drops, highlighting the complex interplay between feature type, model architecture, and resolution in SER tasks.

The steady improvement in performance with increased resolution across CNN, DenseNet, and ResNet models emphasizes their efficacy in leveraging high-resolution data for emotion recognition. Conversely, VGG and Inception models exhibit varied performances, indicating a more nuanced applicability in SER endeavours based on the dataset and feature type context.

This detailed examination illuminates the critical roles of image resolution and feature type in SER model performance, offering a foundation for optimizing input feature representation and model architecture in emotion recognition from speech. It also highlights the potential for future research aimed at refining the selection and application of image-based features in SER.

## Selection

### Model Selection Justification: CNN Family Models

In our efforts to advance Speech Emotion Recognition (SER) technology, selecting the right model architecture is crucial. After extensive evaluations, we have chosen to focus on the CNN family, including CNN, DenseNet, and ResNet, due to their robust performance across various tests. These models have demonstrated remarkable consistency and adaptability across different datasets, particularly in the EMO-DB and RAVDESS datasets. They effectively manage varied data structures—from the straightforward arrangement of EMO-DB to the more complex, actor-wise hierarchical organization in RAVDESS. Their ability to maintain high performance across these diverse data arrangements is indicative of their strong generalization capabilities, which are essential for the varied dataset requirements in SER.

These models are particularly adept at processing Mel-Spectrogram features, essential for capturing the nuanced emotional states depicted in speech. This capability is crucial for datasets like RAVDESS, which features a broad spectrum of emotional expressions across different actors. The superior handling of Mel-Spectrogram inputs by CNN, DenseNet, and ResNet across various resolutions underscores their effectiveness in extracting detailed emotional cues, which is pivotal for accurate emotion recognition.

The architectural flexibility of the CNN family also significantly contributes to their selection. These models support deep and complex network structures that are advantageous for the evolving needs of sophisticated SER tasks. For instance, DenseNet's design promotes efficient feature propagation and reuse across its layers, ensuring that essential information is preserved and utilised effectively throughout the network. This feature is particularly beneficial for SER, where capturing every emotional nuance can greatly enhance recognition accuracy. Similarly, ResNet’s ability to address the vanishing gradient problem allows for the construction of deeper networks that learn complex patterns more effectively, without the degradation of performance over depth.

By incorporating the CNN family into our SER analysis, we are not only utilizing the distinct advantages of each model but also exploring the potential of hybrid architectures and ensemble methods. These approaches can integrate the strengths of CNN, DenseNet, and ResNet to achieve superior performance. This strategy ensures that our SER system is robust, adaptable, and capable of handling the complex nuances of human emotions in speech effectively, thus paving the way for significant advancements in the field.

### Dataset Selection Rationale: EMODB & RAVDESS

In the domain of Speech Emotion Recognition (SER), the choice of datasets is as critical as the selection of the model architecture. For our ongoing research, we have strategically chosen EMO-DB and RAVDESS as our primary datasets. These selections are made based on their distinct characteristics, which together provide a comprehensive overview of the model's capabilities in varied contexts.

**EMO-DB**: This dataset is favoured primarily for its high accuracy and clarity, which make it an excellent benchmark for evaluating model performance and feature extraction capabilities. EMO-DB is known for its straightforward file structure, which allows researchers to focus intensively on model tuning and feature extraction methodologies. The simplicity of the dataset structure reduces the complexity typically involved in navigating through more layered data organizations. This characteristic makes EMO-DB an ideal choice for initial model validations and fine-tuning, providing clear insights into the effectiveness of different architectural nuances and feature handling strategies.

**RAVDESS**: In contrast to EMO-DB, RAVDESS introduces a higher level of complexity with its hierarchical and actor-wise organization. This structure challenges the model to maintain high performance across a more diverse and voluminous dataset. Such complexity is crucial for evaluating the model’s scalability and robustness under varied and challenging conditions. Additionally, RAVDESS is characterized by its diverse emotional expressions, with multiple actors portraying a range of emotions. This diversity offers a comprehensive canvas for assessing the model's ability to generalize across different vocal characteristics and emotional intensities, thus providing a robust test of the model’s adaptability and accuracy in real-world scenarios.

### Optimisation

As we progress in enhancing our Speech Emotion Recognition (SER) systems, several innovative experiments and analytical approaches are planned to refine and advance our models. These initiatives are aimed at both augmenting the datasets and optimizing model performance through a series of structured experimental setups. To assess the impact of image data augmentation on model performance, an experiment will be designed. This will involve a comparative analysis of the baseline CNN performance with and against an augmented dataset version. Techniques such as rotation, scaling, and flipping could be explored to determine their efficacy in enhancing the robustness and generalization capabilities of our models.

An automated hyperparameter optimization process will be implemented, utilizing advanced techniques like Genetic Algorithm optimization. The objective here is to systematically explore and identify the optimal configuration of CNN parameters that maximizes performance specifically on the EMO-DB and RAVDESS datasets. This step is crucial for ensuring our models are not only accurate but also efficient and scalable.

Another experimental avenue involves testing the SER models on a reduced emotion spectrum. Instead of the broader range presented in EMO-DB’s 7 and RAVDESS’s 8 emotion categories, we will explore the model's performance with only 4 classes. This will provide insights into the model's effectiveness in a more generalized approach to emotion recognition, which could be beneficial for applications requiring less granularity in emotional distinction.

# Methodology

## Data Split and Structure

|  |
| --- |
|  |
| *Fig 6:* The Audio dataset being organised into emotion classes subfolders, then converted into images. |

Initially, the audio dataset was meticulously reorganized into class-specific folders. This reorganization involved segregating audio files based on their respective emotional classes such as anger, happiness, sadness, and neutrality. Following the reorganization, the audio data underwent a transformation process where the audio files were converted into image data. This conversion was crucial as it allowed the utilization of convolutional neural networks (CNNs), which are typically more adept at handling image data. By converting audio into a visual format, the dataset leveraged the spatial patterns that CNNs excel at recognizing. It is important to note that each audio file was transformed into a corresponding image, ensuring a one-to-one mapping between the audio inputs and their visual representations. The dataset, now in image form, was split following a standard machine learning practice to aid in both training and validating the models effectively. An 80/20 split was employed, where 80% of the data was used for training the models, allowing them to learn and adapt to the nuances of emotional cues depicted in the images. The remaining 20% served as a test set, for evaluating the model's performance and its ability to generalize on unseen data.

## Audio Feature Extraction and Conversion

|  |
| --- |
|  |
| *Fig 7: Diagram of how audio files are transformed into image, using feature extraction to extract precisely 3 features, and concatenating them towards each of the Red, Green, and Blue channels (RGB) of the image, to create a unique visual representation of that audio, into a single image.* |

The process of audio feature extraction and conversion forms a critical component of the methodology in Speech Emotion Recognition (SER) system. The primary goal here is to transform raw audio data into a format that can be effectively processed by convolutional neural networks (CNNs), which are generally applied to image data. This transformation involves a detailed procedure of extracting significant audio features and encoding them as visual data in the form of RGB images.

### Feature Extraction

The extraction of audio features utilises the Librosa library, a powerful tool for music and audio analysis. Three key types of features are computed from the audio files:

Mel-Frequency Cepstral Coefficients (MFCCs): These features are crucial for capturing the timbral texture of the sound. MFCCs effectively represent the short-term power spectrum of a sound, emphasizing the perceptually important aspects of the audio. They are particularly adept at encapsulating the properties of a voice that are invariant to pitch and time scaling.

Mel-Spectrogram: This feature represents the energy distribution of the sound over time and is computed as a spectrogram with a Mel scale on the frequency axis. The Mel-spectrogram is vital for capturing rhythmic structures and can help in distinguishing between different emotional states based on the intensity and rhythm of speech.

Chroma Features: These are designed to highlight the harmonic and melodic characteristics of the audio. Chroma features capture the essence of music and tone, which are pivotal in identifying and differentiating the emotional content in speech, such as excitement, calmness, or sadness.

### RGB Conversion

Once these audio features are extracted, the next step is to prepare them for CNN processing. This involves several sub-steps:

* **Normalization and Resizing:** Each type of extracted feature undergoes normalization to scale the feature values between 0 and 1. This step is crucial for removing amplitude variations in the audio signal and standardizing the input for consistent processing. Following normalization, the features are resized to a uniform dimension, typically *256x256* pixels. This resizing ensures that each feature fits the expected input size of the CNN, maintaining the integrity and proportionality of the data.
* **Grayscale Conversion:** The normalized and resized features are initially in grayscale format, where each pixel intensity represents a value of the feature. Grayscale images are critical as they maintain the depth and complexity of the features without introducing colour biases that are irrelevant to the audio analysis.
* **Mapping to RGB Channels:** To leverage the power of pre-trained image models, these grayscale images are then mapped onto the RGB channels of a new image. Each feature type is assigned to a specific channel: MFCCs might be assigned to the red channel, Mel-spectrogram to the green, and Chroma to the blue. This method of assignment is strategic; it utilises the CNN's ability to detect patterns from colour channels and applies it to feature differentiation in audio analysis.

The resultant RGB images serve as a fusion of significant audio characteristics represented visually. This innovative approach allows the use of advanced image-processing models, originally designed for visual data, to be effectively applied to the domain of audio analysis. By transforming the audio classification problem into an image classification one, we harness the robust capabilities of CNNs and the extensive developments in image recognition to enhance the accuracy and efficiency of emotion recognition in speech

## Re-Imagined Datasets

|  |  |  |  |
| --- | --- | --- | --- |
| Red | Green | Blue | **OUTPUT** |
| **Red** | Chroma | Mel-Spectrogram | MFCC | *CH\_ME\_MF* |
| **Green** | MFCC | Chroma | Mel-Spectrogram | *MF\_CH\_ME* |
| **Blue** | Mel-Spectrogram | MFCC | Chroma | *ME\_MF\_CH* |
| **OUTPUT** | *CH\_MF\_ME* | *ME\_CH\_MF* | *MF\_ME\_CH* |

|  |
| --- |
| *Fig 8: The 6 different variations in which only 3 features (MFCC, Mel-Spectrogram, and Chroma) can be mapped in certain arrangements to create a unique interpretation and variation of the auditory dataset. The OUTPUT names reflect the orientation in which each feature was mapped into the RGB channels of the image.* |

|  |  |  |
| --- | --- | --- |
| CH\_ME\_MF | MF\_CH\_ME | ME\_CH\_MF |
| A close-up of a blue and pink background  Description automatically generated | A red and green striped background  Description automatically generated | A blue and green striped background  Description automatically generated |
| CH\_MF\_ME | *MF\_ME\_CH* | *ME\_MF\_CH* |
| A green and yellow striped background  Description automatically generated | A red and blue striped background  Description automatically generated with medium confidence | A green and blue striped background  Description automatically generated |
| *Fig 9: A snippet of the same audio file from EMO-DB’s dataset, visualised in all 6 variations of the RGB-Encoded images. All of them convey the same audio file, but depending on the system, results may vary.* | | |

Recognizing the distinct properties and contributions of various audio features, namely Mel-Frequency Cepstral Coefficients (MFCCs), Chroma, and Mel-Spectrogram, a strategic decision was made to manipulate these features across the RGB channels in various combinations. This manipulation resulted in the creation of six different variants of RGB-encoded datasets, each varying the order of the feature maps.

The unique methodology involves assigning the audio features to different colour channels in an RGB image. For example, one variant might encode MFCCs in the red channel, Mel-Spectrogram in the green, and Chroma in the blue. Another variant would shuffle these assignments, thus altering the emphasis placed on each feature in the image processing pipeline of the CNN. This creative reassignment allows each variant dataset to highlight different aspects of the audio data:

Emphasis Variation: By changing which feature is presented in each channel, models may develop a sensitivity to different characteristics of the sound, such as its rhythm, tone, or texture, depending on the channel's prominence in image processing.

Model Training Diversity: This approach allows the same model architecture to be trained on different versions of the data, potentially learning to extract and leverage diverse patterns and insights that are not apparent when trained on a single, static dataset.

Experimental Flexibility: The creation of multiple dataset variants provides a robust framework for experimentation. Researchers can systematically evaluate and compare the performance of models across these datasets, gaining a deeper understanding of how each feature influences the recognition of emotions in speech.

Ultimately, this innovative strategy of dataset variation not only enriches the training environment but also enhances the generalizability and robustness of the models. Each dataset variant can reveal unique insights and performance characteristics, making this an excellent technique for thorough experimentation and analysis in the field of emotion recognition through speech.

## Augmentation

|  |
| --- |
|  |
| *Fig 10:* Using Pre-Trained model, with the Top Layer removed, to create Feature Maps, and clustering them before one-hot encoding and concatenating the original feature maps, with the clustered labels, to create an Augmented Feature Set for an improved training input. |

### Second-Stage Feature Extraction

Another stage of feature extraction takes place, for the new dataset variants. For this purpose, a pre-trained convolutional neural network (CNN) is employed, specifically pre-loaded with weights from ImageNet. The decision to use a pre-trained model stems from the significant advantages it offers in terms of learned feature representations from a vast and varied visual dataset. By setting *include\_top=False*, the network is reconfigured to operate without its final classification layers. This adjustment transforms the network into a sophisticated and ideal feature extraction mechanism.

The output from this adapted CNN then feeds into a *GlobalAveragePooling2D* layer. This layer performs a critical function by reducing the dimensionality of each feature map generated by the convolutional layers to a single vector per map. The action of averaging out the spatial dimensions of the feature maps focuses the subsequent analysis on the most salient features, effectively summarizing the complex data into a more manageable form. This simplification is essential as it retains the most informative parts of the data, thus enhancing the efficiency and performance of the emotion recognition process.

### Clustering

To further refine the model's capacity to discern subtle variations between similar emotional states, a clustering technique is integrated into the feature extraction pipeline. Specifically, *KMeans* clustering is applied to the features extracted by the CNN. This method operates in the high-dimensional space of the features, identifying distinct clusters based on the intrinsic patterns and similarities within the data. Such clustering is particularly beneficial for highlighting nuanced differences between the images that might represent different emotional expressions.

Once clusters are defined, their labels are transformed using one-hot encoding. This encoding converts the cluster labels into a binary format, which is then concatenated with the original feature vectors extracted from the images. This process effectively augments the feature set, providing the subsequent classification model with enriched contextual information. The inclusion of cluster labels as part of the feature set introduces a new dimension of data that the model can use to better differentiate between emotions, potentially improving the accuracy and robustness of the emotion classification.

## Classification Models

In exploring sophisticated architectures for Speech Emotion Recognition (SER), we delve into several prominent convolutional neural network (CNN) models, each offering unique strengths crucial for effectively capturing and classifying emotional cues from RGB-encoded audio features. These architectures—DenseNet121, EfficientNetB0, ResNet50, and VGG16—are employed to enhance the robustness and accuracy of emotion classification, harnessing their complementary strengths to address the diversity of emotions within the datasets.

### DenseNet121

Densely Connected Convolutional Networks (DenseNets) address several challenges encountered in traditional convolutional neural networks (CNNs), such as vanishing gradients, feature reuse, and parameter efficiency (Huang et al., 2017). Introduced by Huang et al. (2017), DenseNets feature a distinctive architecture where each layer connects directly to every subsequent layer, in stark contrast to conventional CNNs where layers connect sequentially.

The cornerstone of DenseNet architecture is its dense connectivity pattern. This setup ensures that each layer receives inputs from all preceding layers, establishing a total of L(L+1)/2 connections for an L-layer network, as opposed to the L connections typical in traditional architectures (Huang et al., 2017). This unique structure enables several critical benefits:

1. **Reduction of the Vanishing Gradient Problem:** Dense connectivity ensures that gradients from the loss function can propagate directly through all layers, mitigating the vanishing gradient issue commonly seen in deep networks.
2. **Enhanced Feature Reuse:** Every layer accesses the feature-maps from all previous layers, which encourages the reuse of features across the network and reduces the need to relearn redundant features (Huang et al., 2017).
3. **Efficient Use of Parameters:** Despite the dense connectivity, DenseNets often require fewer parameters than conventional deep networks because they eliminate the need to relearn redundant features. Each layer in a DenseNet is narrow, containing fewer filters, and contributes only a small set of new feature-maps to the collective knowledge of the network, enhancing parameter efficiency (Huang et al., 2017).

DenseNets have demonstrated superior performance and significant accuracy improvements on several benchmark datasets including CIFAR-10, CIFAR-100, SVHN, and ImageNet, often outperforming other state-of-the-art architectures while maintaining greater parameter efficiency. The robust architecture of DenseNets not only facilitates easier training of deep networks but also ensures computational efficiency and enhanced performance, making them especially suitable for tasks involving visual object recognition (Huang et al., 2017).

### EfficientNetB0

EfficientNetB0, introduced by Tan and Le in 2019, represents a pivotal shift in scaling convolutional neural networks (CNNs) by uniformly adjusting the network's depth, width, and resolution through a compound coefficient. This innovative approach optimises CNNs not only for greater accuracy but also for efficiency, making it highly effective for varied computational environments and applications.

The architecture of EfficientNetB0, as detailed in their seminal paper, involves a meticulous balance of scaling dimensions to enhance performance systematically. At its core, EfficientNetB0 employs a baseline network that uses a compound scaling method to proportionately increase the dimensions of depth, width, and resolution, based on a set compound coefficient (Tan and Le, 2019). This is fundamentally different from traditional scaling methods that independently adjust these factors, often leading to inefficient performance scaling.

EfficientNetB0's structure includes layers such as convolutional layers, batch normalization, and swish activation, which collectively aim to maintain a high efficiency. The model begins with an input layer followed by various convolutional and pooling layers, integrating depth wise separable convolutions that reduce computational cost while capturing intricate features in the input data.

The effectiveness of EfficientNetB0 is further underscored by its performance on benchmark datasets like ImageNet, where it achieves state-of-the-art accuracy with significantly fewer parameters and lower computational requirements compared to other models like GPipe and traditional ResNets. This efficiency is attributed to the balanced scaling of model dimensions, which ensures that each aspect contributes optimally to the network's learning capacity.

In conclusion, EfficientNetB0's development showcases a strategic refinement of network scaling, emphasizing a balanced increase across all dimensions—depth, width, and resolution—to achieve unprecedented levels of efficiency and effectiveness in CNN architectures (Tan, M. and Le, Q., 2019). This approach not only advances the performance metrics but also sets a new standard for developing lightweight yet powerful neural networks suitable for diverse applications, from mobile devices to high-end servers.

### ResNet50

ResNet50 is an influential deep learning model introduced by He et al. (2016) in their paper "Deep Residual Learning for Image Recognition". The model is built on the concept of residual learning, which helps to alleviate the issues associated with training very deep neural networks. The architecture utilises residual blocks where shortcut connections, or "identity mappings", are used. These connections skip one or more layers and perform element-wise addition, allowing gradients to flow through the network without attenuation during training. This approach directly addresses the vanishing gradients problem, enabling the successful training of networks with substantially increased depth and complexity.

ResNet50 specifically uses a 50-layer deep network that comprises several stacked residual blocks, each refining the feature maps while maintaining the underlying information through shortcuts. This deep architecture allows it to learn rich and complex representations of the data, making it exceptionally capable of handling a wide variety of computer vision tasks, ranging from image classification to object detection.

Each residual block in ResNet50 typically contains three layers: a 1x1 convolution that reduces dimensionality, a 3x3 convolution that processes features, and another 1x1 convolution that restores dimensions, forming a bottleneck structure. This design is not only computationally efficient but also leverages the benefits of deep networks in learning features at various levels of abstraction, thereby improving performance on tasks requiring recognition of intricate patterns.

In summary, ResNet50's design incorporates shortcuts that make deeper networks feasible and effective by enabling the training process to converge faster and more reliably. This model set a new standard in deep learning architectures by demonstrating that increasing network depth can significantly enhance performance when residual connections are used to sustain the flow of gradients throughout the network (He et al., 2016).

### VGG16

Developed by Karen Simonyan and Andrew Zisserman in 2014, VGG16 is a cornerstone model in the convolutional neural networks (CNNs) landscape, renowned for its simplicity and depth which contribute significantly to its effectiveness in large-scale image recognition tasks. The VGG16 architecture is characterized by its uniform use of 3x3 convolutional filters throughout its depth, interspersed with max pooling layers to reduce dimensionality at various stages. This uniformity simplifies the learning process, allowing the model to efficiently learn patterns from images.

One of the standout features of VGG16 is its depth—16 layers deep, hence its name. It incorporates several convolutional layers followed by three fully-connected layers, which culminate in a final softmax classification layer. This depth and structure enable the network to capture complex features at various levels, making VGG16 highly effective for image classification tasks, including facial recognition and image segmentation.

The consistent filter size and the depth of the network make it a robust choice for feature extraction, even in complex scenarios. This capability has led to VGG16 being employed as a pre-trained model for various applications beyond simple classification, serving as a feature extractor for more complicated tasks such as object detection and more.

Despite its many layers, VGG16's architecture is straightforward, which makes it easy to implement and adapt for various convolutional neural network applications. This ease of use, combined with its strong performance on visual tasks, has ensured its popularity continues long after its initial development.

## Optimisation

Genetic algorithms (GAs) are a subset of evolutionary algorithms used to solve optimization problems through techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover. In the context of optimizing neural network hyperparameters, GAs provide a robust methodology to search through the hyperparameter space in an efficient manner, potentially outperforming traditional grid or random search methods by focusing on the exploration of promising areas of the space.

### Hyperparameters

|  |
| --- |
| def initialize\_population(pop\_size):  population = []  for \_ in range(pop\_size):  individual = {  'learning\_rate': 10 \*\* np.random.uniform(-5, -1),  'batch\_size': np.random.choice([16, 32, 64, 128]),  'dense\_neurons': np.random.choice([64, 128, 256, 512]),  'activation': np.random.choice(['relu', 'tanh', 'sigmoid', 'leaky\_relu', 'elu']),  'dropout\_rate': np.random.uniform(0.01, 0.5),  'n\_clusters': np.random.randint(2, 20),  'unfreeze': random.choice([True, False]), # Randomly decide to unfreeze or not  'layers\_to\_unfreeze': np.random.randint(0, 5) # Random number of layers to unfreeze  }  population.append(individual)  return population  POPULATION\_SIZE = 10 NUM\_GENERATIONS = 20 NUM\_PARENTS = 5 |
| *Fig 11: Snippet of the Genetic Algorithm’s code, where the hyperparameters are being settled and ready to be embedded into the population and genetically be tweaked by evolution over multiple generations, by inheriting the best performing attributes (values for each hyperparameter) thus naturally selecting the best optimised model after 20 Generation. Each Generation will contain a POPULATION size of 10, in which they will create the next generation of 10 models, each of them inheriting from 5 previous generational models.* |

Hyperparameters in neural networks are parameters whose values are set before the learning process begins. These parameters play a crucial role in controlling the behaviour of the training algorithm and the structure of the neural network itself. Typical hyperparameters include the learning rate, batch size, number of neurons in dense layers, activation functions, dropout rate, and others.

For models where weights are initially frozen, the genetic algorithm in this setup does not include *unfreeze* or ***layers\_to\_unfreeze*** as hyperparameters. Freezing layers is a technique often used to fine-tune pre-trained models, where the initial layers retain their trained weights (and are thus "frozen") and only the upper layers are trained to adapt to new data. The GA optimises other hyperparameters while keeping these layers frozen, streamlining the adaptation process without overwhelming computational costs or risk of overfitting.

### Layers & Weights

In neural networks, layers are the basic units of structure, and weights are the parameters adjusted during training. The initial choice of how many layers to use, and how many neurons each layer contains, can dramatically affect network performance. Weights, once initialized, are updated through backpropagation as the network learns from exposure to training data. For genetic algorithms, each individual in the population might represent a different configuration of these layers and weights. By simulating a process of evolution, the algorithm iteratively selects the best-performing configurations and combines their characteristics (crossover), introducing random modifications (mutations) to explore new parts of the hyperparameter space.

### Genetic Algorithm

The genetic algorithm starts by initializing a population of individuals, where each individual represents a potential solution to the optimization problem—in this case, a set of hyperparameters for the neural network. The *initialize\_population* function shown generates a diverse set of initial solutions based on predefined ranges for each hyperparameter. Each individual in the population is then evaluated based on a fitness function, which, in machine learning applications, is often related to the model's performance on a validation set. The best-performing individuals are selected as parents to produce the next generation. Offspring are created through crossover—a process where parent individuals combine their features (hyperparameters) to create new offspring—and mutation, where small, random changes are introduced to offspring hyperparameters to maintain genetic diversity within the population.This process repeats over several generations (as defined by ***NUM\_GENERATIONS***), with the algorithm aiming to improve the fitness of the best individual in each successive generation. The final outcome is a set of optimised hyperparameters that ideally offer superior performance compared to randomly selected or manually tuned alternatives. This approach not only saves significant amounts of time and computational resources but also uncovers high-performing hyperparameter sets that might not be intuitive or otherwise discovered through conventional methods.

# Results

## Frozen Layers

|  |  |
| --- | --- |
| **EMODB** | **RAVDESS** |
|  |  |
|  |  |
| ***Fig 12:*** *The RGB-Encoded dataset results of the Pre-Trained models with Frozen Layers, between EMODB and the RAVDESS dataset, during ALL CLASS classification and 4 CLASS classification* | |

Among the architectures evaluated, EfficientNet demonstrated exceptional robustness and adaptability, consistently achieving high performance metrics in both datasets. For instance, its *MF\_ME\_CH* configuration achieved an impressive Macro Average of *75%* in EMO-DB and *68%* in RAVDESS, suggesting its superior capability in capturing nuanced emotional expressions through effective feature integration. This model's performance indicates a balanced handling of the datasets' characteristics, where the varying emotional intensity and diversity between EMO-DB and RAVDESS were well-managed.

Conversely, aside from the baseline CNN model, DenseNet, while generally showing good results, lagged slightly in RAVDESS with its top configuration (*MF\_ME\_CH*) only reaching a Macro Average of *63%*. This drop in performance could be attributed to the model's sensitivity to dataset-specific features, which may have been more pronounced in RAVDESS’s acoustically and emotionally complex samples. In comparison, other models like ResNet and VGG showcased better consistency between the datasets. Particularly, ResNet’s *MF\_ME\_CH* configuration displayed strong feature extraction capabilities with Macro Averages of *76%* in EMO-DB and *69%* in RAVDESS. VGG excelled in integrating complex features, especially in the *ME\_MF\_CH* combination, which recorded the highest Macro Average of *80%* in EMO-DB, though it saw a moderate decrease in RAVDESS. These results underline the importance of choosing the right model and feature configuration that aligns with the specific challenges and characteristics of the dataset at hand.

Focusing on four primary emotions (Angry, Happy, Neutral, Sad), the models revealed more pronounced distinctions in their classification capabilities. In this streamlined evaluation, DenseNet emerged as the standout model in the EMO-DB dataset, where its *CH\_ME\_MF* configuration soared to a Macro Average of *95%*, reflecting not only high accuracy but also an exceptional ability to differentiate between closely related emotional states. Its performance in RAVDESS, although slightly reduced, remained robust, reinforcing DenseNet's strong generalization across different emotional content. EfficientNet and ResNet also exhibited remarkable performance, with EfficientNet’s *MF\_ME\_CH* configuration peaking at a Macro Average of *93%* in EMO-DB and *78%* in RAVDESS, affirming its effective handling of feature arrangements critical for emotion recognition.

In a notable contrast, VGG, while slightly underperforming in the all-class analysis, demonstrated remarkable prowess in the four-class scenario, achieving near-perfect Macro Averages in EMO-DB with configurations like *ME\_CH\_MF* and *MF\_ME\_CH* both achieving a Macro Average of *97%*. This significant performance uplift highlights VGG’s capacity to excel in more concentrated classification tasks where the emotional categories are less varied but require high precision. For RAVDESS, however, while VGG’s performance was commendable, it did not surpass the effectiveness observed in the EMO-DB dataset, indicating possible variations in model performance driven by dataset-specific nuances.

|  |  |
| --- | --- |
| **EMODB** | |
| A graph of blue and white bars  Description automatically generated | A graph of blue and white bars  Description automatically generated |
| **RAVDESS** | |
| A graph of blue and white bars  Description automatically generated | A graph of blue and white bars  Description automatically generated |
| ***Fig 13:*** *Every model’s best performing dataset variant, using frozen weights using all class classification and 4 class classification* | |

## Unfrozen Layers

|  |  |
| --- | --- |
| **EMODB** | **RAVDESS** |
|  |  |
|  |  |
| ***Fig 14:*** *The RGB-Encoded dataset results of the Pre-Trained models with Unrozen Layers, between EMODB and the RAVDESS dataset, during ALL CLASS classification and 4 CLASS classification* | |

The unfrozen layers model comparison between CNN, DenseNet, EfficientNet, ResNet, and VGG across the EMO-DB and RAVDESS datasets reveals varied performance, underscoring the significant impact of architectural complexity and feature extraction depth on classification effectiveness. Notably, DenseNet and EfficientNet consistently demonstrate superior performance, with DenseNet achieving a high of *77%* Macro Average in EMO-DB with the *CH\_ME\_MF* configuration and EfficientNet matching this performance in RAVDESS with the *MF\_ME\_CH* setup.

Comparatively, CNN acts as a baseline with moderate performance, highlighting the advantages of more complex models. For example, CNN's best Macro Average in EMO-DB was only *62%*, significantly lower than DenseNet's and EfficientNet's. The results emphasize a clear trend: as model complexity increases, so does the ability to handle the nuanced features of speech emotion, especially when the layers are unfrozen and can adapt more dynamically to the training data.

ResNet and VGG also show strong performances but do not consistently reach the heights of DenseNet and EfficientNet. However, ResNet's *MF\_ME\_CH* configuration in EMO-DB stands out with a *78%* Macro Average, suggesting its potential in specific settings. VGG's best performance, showing a *78%* Macro Average in EMO-DB and a *63%* in RAVDESS with the *CH\_ME\_MF* and *MF\_ME\_CH* configurations respectively, indicates its competence but also highlights its variability across datasets.

In the more focused four-class scenario, where only primary emotions (Angry, Happy, Neutral, Sad) are considered, the models exhibit an interesting shift in performance dynamics. DenseNet and VGG particularly shine in EMO-DB, with both reaching exceptionally high Macro Averages of *94%* and *97%*, respectively, in their optimal configurations. These results illustrate the models' acute sensitivity and accuracy in distinguishing between these more broadly defined emotional states.

EfficientNet and ResNet also perform admirably, with EfficientNet peaking at a *92%* Macro Average in EMO-DB using the *MF\_ME\_CH* configuration and ResNet showing a robust *92%* in the same dataset with the *CH\_ME\_MF* and *CH\_MF\_ME* setups. Their performances, while slightly trailing behind DenseNet and VGG, still demonstrate significant efficacy in emotion classification.

RAVDESS, known for its complexity and nuanced emotional content, presents a greater challenge, as evidenced by generally lower performance metrics across all models. However, DenseNet and EfficientNet manage to adapt better than others, with DenseNet achieving a *77%* Macro Average and EfficientNet matching this performance in the *MF\_ME\_CH* configuration. This adaptation indicates their potential utility in more complex emotional recognition tasks where nuanced distinctions are critical.

|  |  |
| --- | --- |
| **EMODB** | |
| A graph of different colored bars  Description automatically generated | A graph of different colors  Description automatically generated |
| **RAVDESS** | |
| A graph of different colors  Description automatically generated | A graph of different colored bars  Description automatically generated |
| ***Fig 15:*** *Every model’s best performing dataset variant, using frozen weights using all class classification and 4 class classification* | |

## Per-Variant

|  |  |
| --- | --- |
| **CH\_ME\_MF** | |
| A graph of different colored bars  Description automatically generated | A graph of different colored bars  Description automatically generated |
| **CH\_MF\_ME** | |
| A graph of different colored bars  Description automatically generated | A graph of different colored bars  Description automatically generated |
| **ME\_CH\_MF** | |
| A graph of different colored bars  Description automatically generated | A graph of different colored bars  Description automatically generated |
| **ME\_MF\_CH** | |
| A graph of different colored bars  Description automatically generated | A graph of different colored bars  Description automatically generated |
| **MF\_CH\_ME** | |
| A graph of different colored bars  Description automatically generated | A graph of different colored bars  Description automatically generated |
| **MF\_ME\_CH** | |
| A graph of different colored bars  Description automatically generated | A graph of different colored bars  Description automatically generated |
| ***Fig 16:*** *Each dataset variant’s performance across every scenario for all class classification and 4 class classification* | |

Configurations like ***CH\_ME\_MF*** and ***ME\_MF\_CH*** leverage the model's inherent visual processing capabilities to different extents, based on the feature placed in each channel. Typically, placing Chroma in the red channel and MFCC in the blue channel demonstrates a nuanced understanding of the model's sensitivity to channel priority, where Chroma's harmonic content and MFCC's timbral information are differently emphasized. Models like DenseNet and EfficientNet exhibit robust performance across these configurations, suggesting their architectures are well-suited to integrate and interpret the complex inter-channel relationships of these encoded features. In contrast, models like VGG, which are highly sensitive to textural details, show varying results that underscore the critical role of feature placement within the RGB spectrum.

The all-class approach reveals that while some configurations yield high performance consistently, others do not, highlighting the importance of strategic feature placement. For example, placing the Mel-Spectrogram in the green channel often results in better performance across models, possibly due to the channel's higher sensitivity to the dynamic range, which is crucial for capturing the energy fluctuations in emotional speech. This observation suggests that the effectiveness of the RGB-encoding method depends significantly on aligning the models' architectural strengths with the characteristics of the features placed in each RGB channel. Moreover, the variability in performance across different datasets like EMO-DB and RAVDESS illustrates the method's adaptability to diverse acoustic and emotional environments, with certain configurations better handling the complexities associated with different types of emotional data.

Focusing on four primary emotions—Angry, Happy, Neutral, Sad—the RGB-encoded approach takes on added significance in delineating clear emotional boundaries within the data. This focused scenario allows for a deeper analysis of how specific feature arrangements impact model performance in recognizing more distinctly categorized emotional states. In this context, DenseNet and VGG particularly stand out, achieving exceptionally high accuracy and Macro Averages, which indicate their effective use of textural and pattern recognition capabilities to discern subtle differences between these primary emotions. The superior performance of configurations like *CH\_ME\_MF* in DenseNet and *MF\_ME\_CH* in VGG within the EMO-DB dataset suggests that these models are exceptionally proficient at leveraging the detailed textural information provided by the Mel-Spectrogram and MFCC when these features are prioritized in the visual spectrum.

# Discussion

## Interpretation and Challenges

The strategic deployment of RGB-encoded audio datasets offers a significant transformation in the methodology of speech emotion recognition. By encoding audio features like MFCC, Mel-Spectrogram, and Chroma into RGB images, these models utilise the inherent strengths of convolutional neural networks in visual data interpretation, which facilitates enhanced emotion recognition. This technique not only enables the use of sophisticated visual processing models like DenseNet and VGG but also aligns with the compact and lightweight nature of image files compared to audio files, offering efficiencies in storage and computational processing. However, the analysis reveals that different models exhibit varying levels of adaptability to this encoding method, particularly when comparing performance across the EMO-DB and RAVDESS datasets. For instance, while VGG shows superior performance in the EMO-DB dataset, it underperforms in the RAVDESS dataset. This inconsistency could be attributed to the complexity of RAVDESS, which includes both speech and singing audio. The dual nature of RAVDESS's audio content introduces a significant challenge in emotion classification, as models trained on both may struggle to effectively distinguish between spoken and sung expressions, highlighting a potential area for model refinement and training strategy adjustment.

## Confusion of Emotional Categories and Future Directions

A notable observation from the All-Class metrics in EMO-DB is the frequent confusion between the emotions Angry and Happy. This confusion points to a potential overlap in the acoustic features of these emotions as captured and processed by the models. To address this and similar challenges, it is suggested that future research in speech emotion recognition may benefit from considering emotions not as distinct categories but rather as a spectrum. This approach would involve main emotions serving as primary classes and other ambiguous or relative emotions as sub-classes, potentially represented in multi-dimensional vectors. This dimensional approach could necessitate more complex model architectures capable of interpreting these nuanced classifications.

## Performance Insights and Implications

VGG's inconsistent performance across datasets, excelling in EMO-DB but lagging in RAVDESS, underscores the influence of dataset characteristics on model efficacy. This disparity suggests that while VGG is well-suited to datasets with clearer, more distinct emotional expressions, its performance diminishes in datasets with more complex or blended emotional content. On the other hand, models like DenseNet and EfficientNet demonstrate more consistent robustness across both datasets, indicating their ability to handle a variety of emotional complexities more effectively. The underperformance of all models in RAVDESS relative to EMO-DB further suggests that audio-only datasets may be insufficient for advanced speech emotion recognition tasks. Incorporating visual or contextual data could significantly enhance the models' ability to accurately classify emotions by providing additional layers of emotional cues beyond what audio alone can offer.

# Conclusion

The investigation into RGB-encoded audio features for speech emotion recognition has unequivocally demonstrated its merits, marking a significant step forward in harnessing the full potential of AI in understanding human emotions. This study has validated the efficacy of this innovative approach, showcasing how transforming audio signals into a visual format can leverage the advanced capabilities of convolutional neural networks (CNNs) tailored for image processing.

## Advantages

**Enhanced Model Performance**: The conversion of audio features—MFCC, Mel-Spectrogram, and Chroma—into RGB images aligns with the inherent strengths of CNNs in detecting patterns and textures within visual data. This alignment enables models to effectively parse complex emotional nuances from sound, which traditional audio-only models might overlook. For instance, the precision with which models like VGG and DenseNet, when adapted to this RGB framework, could distinguish between varied emotional states in datasets underscores the method's utility in enhancing classification accuracy.

**Data Efficiency**: One of the salient benefits of RGB-encoding is the reduction in data complexity. Audio files, typically larger and more complex, can be transformed into a format that is inherently more compact and efficient to process. This efficiency not only streamlines the computational demands but also simplifies the data pipeline, reducing the overhead associated with data storage and processing.

**Scalability and Flexibility**: The RGB-encoded approach introduces scalability in handling diverse datasets. By visualizing audio features, researchers and developers can apply a broader range of image processing techniques that are mature and well-developed. This flexibility is crucial for tailoring solutions to specific emotional recognition tasks, which may vary widely across different languages, dialects, and cultural contexts.

**Improved Feature Utilization**: Placing different audio features across the RGB channels allows distinct aspects of the data to be emphasized or attenuated according to the emotional recognition needs. For instance, emphasizing the Mel-Spectrogram in the green channel takes advantage of the channel's sensitivity to dynamic ranges, crucial for capturing the intensity and subtlety of emotional expressions. This strategic placement ensures that models can focus on the most informative features, enhancing overall recognition accuracy.

## Potential Weaknesses

While the advantages of RGB-encoded audio features for speech emotion recognition are compelling, it's crucial to acknowledge potential weaknesses inherent in this approach. One significant concern is the risk of information loss during the transformation process from audio signals to RGB images. Critical auditory details that are vital for emotion recognition might not be visually represented with the same level of precision or might be interpreted differently by image-based models. This transformation could lead to a misalignment between the models’ processing capabilities and the subtleties of human emotional expression captured in pure audio form.

Furthermore, the reliance on RGB encoding assumes a uniform effectiveness across different emotional categories, which may not hold true in practice. Emotional nuances that are distinctly auditory in nature, such as intonation and pitch, could be underrepresented in visual formats. This method's effectiveness could also vary significantly across languages and dialects, which encode emotional information in diverse auditory signatures not always amenable to visual representation.

Moreover, the encoding process requires careful calibration and tuning to ensure that no critical information is overlooked or misrepresented. Ensuring the fidelity of emotional data through RGB channels requires sophisticated understanding and adjustments, which could complicate the training and deployment of models, particularly in diverse real-world applications where scalability and adaptability are paramount.

## Future Work

The findings from this study suggest a robust framework for future explorations in the domain of emotion recognition. The success of RGB-encoded audio data in improving model responsiveness and accuracy presents a compelling case for its adoption in more advanced AI systems. Future research could explore integrating multimodal data sources, combining RGB-encoded audio with real-time video or physiological sensors, to create even more nuanced and context-aware emotion recognition systems.

Moreover, further advancements in hardware and software that support higher-resolution and higher-dimensional data representations could unlock new potentials in this field. For instance, exploring beyond the standard 8-bit image format to 16-bit or 32-bit could allow for finer gradations in feature representation, which would enable even more precise model training and emotion classification.

In summary, the strategic use of RGB-encoded audio features for speech emotion recognition not only aligns with current technological trends but also sets a new standard for the field. It encourages a shift from traditional audio signal processing to more integrated, visually oriented AI systems that promise higher accuracy and efficiency in emotion recognition tasks. This study not only highlights the effectiveness of this approach but also points the way forward for its broader application and development, ensuring that as technology evolves, so too does our capacity to understand and interpret human emotions with greater depth and fidelity.

## Final Thoughts

The exploration of RGB-encoded audio features for speech emotion recognition as demonstrated in this research not only emphasises the innovative integration of visual and auditory data but also illustrates the transformative potential of artificial intelligence in understanding and interpreting human emotions. By converting audio into RGB images, this approach capitalizes on the sophisticated pattern recognition capabilities of convolutional neural networks, paving the way for more nuanced and accurate emotion analysis. Despite the inherent challenges such as potential information loss and the need for precise calibration, the benefits of enhanced model performance, data efficiency, scalability, and improved feature utilization, largely outweigh these concerns. As we advance, it is imperative to refine these methodologies, ensuring they are robust enough to handle the complex variances across linguistic and cultural spectrums. The promising results obtained advocate for broader implementations and continual enhancements in AI technologies, which will inevitably increase the effectiveness and applicability of emotion recognition systems. This study not only propels the field forward but also inspires ongoing innovation in AI to better serve and understand the rich tapestry of human expressions and interactions.

**References:**

* + Wani, T.M., Gunawan, T.S., Qadri, S.A.A., Kartiwi, M. and Ambikairajah, E., 2021. A comprehensive review of speech emotion recognition systems. IEEE access, 9, pp.47795-47814.
  + Kerkeni, L., Serrestou, Y., Mbarki, M., Raoof, K., Mahjoub, M.A. and Cleder, C., 2019. Automatic speech emotion recognition using machine learning.
  + Swain, M., Routray, A. and Kabisatpathy, P., 2018. Databases, features and classifiers for speech emotion recognition: a review. International Journal of Speech Technology, 21, pp.93-120.
  + Fayek, H.M., Lech, M. and Cavedon, L., 2017. Evaluating deep learning architectures for speech emotion recognition. Neural Networks, 92, pp.60-68.
  + Pandey, S.K., Shekhawat, H.S. and Prasanna, S.M., 2019, April. Deep learning techniques for speech emotion recognition: A review. In 2019 29th International Conference Radioelektronika (RADIOELEKTRONIKA) (pp. 1-6). IEEE.
  + Aouani, H. and Ayed, Y.B., 2020. Speech emotion recognition with deep learning. Procedia Computer Science, 176, pp.251-260.
  + Lin, Y.L. and Wei, G., 2005, August. Speech emotion recognition based on HMM and SVM. In 2005 international conference on machine learning and cybernetics (Vol. 8, pp. 4898-4901). IEEE.
  + Anusha, R., Subhashini, P., Jyothi, D., Harshitha, P., Sushma, J. and Mukesh, N., 2021, June. Speech emotion recognition using machine learning. In 2021 5th International Conference on Trends in Electronics and Informatics (ICOEI) (pp. 1608-1612). IEEE.
  + Cao, H., Cooper, D.G., Keutmann, M.K., Gur, R.C., Nenkova, A. and Verma, R., 2014. Crema-d: Crowd-sourced emotional multimodal actors dataset. IEEE transactions on affective computing, 5(4), pp.377-390.
  + Burkhardt, F., Paeschke, A., Rolfes, M., Sendlmeier, W.F. and Weiss, B., 2005, September. A database of German emotional speech. In Interspeech (Vol. 5, pp. 1517-1520).
  + Livingstone, S.R. and Russo, F.A., 2018. The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in North American English. PloS one, 13(5), p.e0196391.
  + Jackson, P. and Haq, S. (2014) *Surrey Audio-Visual Expressed Emotion (SAVEE) database*, *Surrey Audio-visual expressed emotion (SAVEE) database*. Available at: http://kahlan.eps.surrey.ac.uk/savee/. <https://openresearch.surrey.ac.uk/esploro/outputs/journalArticle/Surrey-audio-visual-expressed-emotion-savee-database/99635364402346>
  + Huang, G., Liu, Z., Van Der Maaten, L. and Weinberger, K.Q., 2017. Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4700-4708).
  + Tan, M. and Le, Q., 2019, May. Efficientnet: Rethinking model scaling for convolutional neural networks. In International conference on machine learning (pp. 6105-6114). PMLR.
  + He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).
  + Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
  + Latif, S., Rana, R., Khalifa, S., Jurdak, R. and Schuller, B.W., 2020. Deep architecture enhancing robustness to noise, adversarial attacks, and cross-corpus setting for speech emotion recognition. arXiv preprint arXiv:2005.08453.
  + Ottl, S., Amiriparian, S., Gerczuk, M., Karas, V. and Schuller, B., 2020, October. Group-level speech emotion recognition utilising deep spectrum features. In Proceedings of the 2020 International Conference on Multimodal Interaction (pp. 821-826).
  + Chakhtouna, A., Sekkate, S. and Adib, A., 2022, October. Speech emotion recognition using pre-trained and fine-tuned transfer learning approaches. In The Proceedings of the International Conference on Smart City Applications (pp. 365-374). Cham: Springer International Publishing.

**Appendices:**

|  |  |
| --- | --- |
| **DenseNet121 Summary** | **EfficietNetB0 Summary** |
| |  |  |  |  | | --- | --- | --- | --- | | **Layer (type)** | **Output Shape** | **Parameters** | **Connected to** | |  | | **input\_1** | **(InputLayer) [(None, 224, 224, 3** | **0** | **[]** |  | | **zero\_padding2d** | **(ZeroPadding2D) (None, 230, 230, 3)** | **0** | **['input\_1[0][0]']** |  | | **conv1/conv** | **(Conv2D) (None, 112, 112, 64** | **9408** | **['zero\_padding2d[0][0]']** |  | | **conv1/bn** | **(BatchNormalization) (None, 112, 112, 64** | **256** | **['conv1/conv[0][0]']** |  | | **conv1/relu** | **(Activation) (None, 112, 112, 64** | **0** | **['conv1/bn[0][0]']** |  | | **zero\_padding2d\_1** | **(ZeroPadding2 (None, 114, 114, 64** | **0** | **['conv1/relu[0][0]']** |  | | **pool1** | **(MaxPooling2D) (None, 56, 56, 64)** | **0** | **['zero\_padding2d\_1[0][0]']** |  | | **conv2\_block1\_0\_bn** | **(BatchNormal (None, 56, 56, 64)** | **256** | **['pool1[0][0]']** |  | | **conv2\_block1\_0\_relu** | **(Activatio (None, 56, 56, 64)** | **0** | **['conv2\_block1\_0\_bn[0][0]']** |  | | **conv2\_block1\_1\_conv** | **(Conv2D) (None, 56, 56, 128)** | **8192** | **['conv2\_block1\_0\_relu[0][0]']** |  | | **conv2\_block1\_1\_bn** | **(BatchNormal (None, 56, 56, 128)** | **512** | **['conv2\_block1\_1\_conv[0][0]']** |  | | **conv2\_block1\_1\_relu** | **(Activatio (None, 56, 56, 128)** | **0** | **['conv2\_block1\_1\_bn[0][0]']** |  | | **conv2\_block1\_2\_conv** | **(Conv2D) (None, 56, 56, 32)** | **36864** | **['conv2\_block1\_1\_relu[0][0]']** |  | | **conv2\_block1\_concat** | **(Concatena (None, 56, 56, 96)** | **0** | **['pool1[0][0]',** |  | | **conv2\_block2\_0\_bn** | **(BatchNormal (None, 56, 56, 96)** | **384** | **['conv2\_block1\_concat[0][0]']** |  | | **conv2\_block2\_0\_relu** | **(Activatio (None, 56, 56, 96)** | **0** | **['conv2\_block2\_0\_bn[0][0]']** |  | | **conv2\_block2\_1\_conv** | **(Conv2D) (None, 56, 56, 128)** | **12288** | **['conv2\_block2\_0\_relu[0][0]']** |  | | **conv2\_block2\_1\_bn** | **(BatchNormal (None, 56, 56, 128)** | **512** | **['conv2\_block2\_1\_conv[0][0]']** |  | | **conv2\_block2\_1\_relu** | **(Activatio (None, 56, 56, 128)** | **0** | **['conv2\_block2\_1\_bn[0][0]']** |  | | **conv2\_block2\_2\_conv** | **(Conv2D) (None, 56, 56, 32)** | **36864** | **['conv2\_block2\_1\_relu[0][0]']** |  | | **conv2\_block2\_concat** | **(Concatena (None, 56, 56, 128)** | **0** | **['conv2\_block1\_concat[0][0]',** |  | | **conv2\_block3\_0\_bn** | **(BatchNormal (None, 56, 56, 128)** | **512** | **['conv2\_block2\_concat[0][0]']** |  | | **conv2\_block3\_0\_relu** | **(Activatio (None, 56, 56, 128)** | **0** | **['conv2\_block3\_0\_bn[0][0]']** |  | | **conv2\_block3\_1\_conv** | **(Conv2D) (None, 56, 56, 128)** | **16384** | **['conv2\_block3\_0\_relu[0][0]']** |  | | **conv2\_block3\_1\_bn** | **(BatchNormal (None, 56, 56, 128)** | **512** | **['conv2\_block3\_1\_conv[0][0]']** |  | | **conv2\_block3\_1\_relu** | **(Activatio (None, 56, 56, 128)** | **0** | **['conv2\_block3\_1\_bn[0][0]']** |  | | **conv2\_block3\_2\_conv** | **(Conv2D) (None, 56, 56, 32)** | **36864** | **['conv2\_block3\_1\_relu[0][0]']** |  | | **conv2\_block3\_concat** | **(Concatena (None, 56, 56, 160)** | **0** | **['conv2\_block2\_concat[0][0]',** |  | | **conv2\_block4\_0\_bn** | **(BatchNormal (None, 56, 56, 160)** | **640** | **['conv2\_block3\_concat[0][0]']** |  | | **conv2\_block4\_0\_relu** | **(Activatio (None, 56, 56, 160)** | **0** | **['conv2\_block4\_0\_bn[0][0]']** |  | | **conv2\_block4\_1\_conv** | **(Conv2D) (None, 56, 56, 128)** | **20480** | **['conv2\_block4\_0\_relu[0][0]']** |  | | **conv2\_block4\_1\_bn** | **(BatchNormal (None, 56, 56, 128)** | **512** | **['conv2\_block4\_1\_conv[0][0]']** |  | | **conv2\_block4\_1\_relu** | **(Activatio (None, 56, 56, 128)** | **0** | **['conv2\_block4\_1\_bn[0][0]']** |  | | **conv2\_block4\_2\_conv** | **(Conv2D) (None, 56, 56, 32)** | **36864** | **['conv2\_block4\_1\_relu[0][0]']** |  | | **conv2\_block4\_concat** | **(Concatena (None, 56, 56, 192)** | **0** | **['conv2\_block3\_concat[0][0]',** |  | | **conv2\_block5\_0\_bn** | **(BatchNormal (None, 56, 56, 192)** | **768** | **['conv2\_block4\_concat[0][0]']** |  | | **conv2\_block5\_0\_relu** | **(Activatio (None, 56, 56, 192)** | **0** | **['conv2\_block5\_0\_bn[0][0]']** |  | | **conv2\_block5\_1\_conv** | **(Conv2D) (None, 56, 56, 128)** | **24576** | **['conv2\_block5\_0\_relu[0][0]']** |  | | **conv2\_block5\_1\_bn** | **(BatchNormal (None, 56, 56, 128)** | **512** | **['conv2\_block5\_1\_conv[0][0]']** |  | | **conv2\_block5\_1\_relu** | **(Activatio (None, 56, 56, 128)** | **0** | **['conv2\_block5\_1\_bn[0][0]']** |  | | **conv2\_block5\_2\_conv** | **(Conv2D) (None, 56, 56, 32)** | **36864** | **['conv2\_block5\_1\_relu[0][0]']** |  | | **conv2\_block5\_concat** | **(Concatena (None, 56, 56, 224)** | **0** | **['conv2\_block4\_concat[0][0]',** |  | | **conv2\_block6\_0\_bn** | **(BatchNormal (None, 56, 56, 224)** | **896** | **['conv2\_block5\_concat[0][0]']** |  | | **conv2\_block6\_0\_relu** | **(Activatio (None, 56, 56, 224)** | **0** | **['conv2\_block6\_0\_bn[0][0]']** |  | | **conv2\_block6\_1\_conv** | **(Conv2D) (None, 56, 56, 128)** | **28672** | **['conv2\_block6\_0\_relu[0][0]']** |  | | **conv2\_block6\_1\_bn** | **(BatchNormal (None, 56, 56, 128)** | **512** | **['conv2\_block6\_1\_conv[0][0]']** |  | | **conv2\_block6\_1\_relu** | **(Activatio (None, 56, 56, 128)** | **0** | **['conv2\_block6\_1\_bn[0][0]']** |  | | **conv2\_block6\_2\_conv** | **(Conv2D) (None, 56, 56, 32)** | **36864** | **['conv2\_block6\_1\_relu[0][0]']** |  | | **conv2\_block6\_concat** | **(Concatena (None, 56, 56, 256)** | **0** | **['conv2\_block5\_concat[0][0]',** |  | | **pool2\_bn** | **(BatchNormalization) (None, 56, 56, 256)** | **1024** | **['conv2\_block6\_concat[0][0]']** |  | | **pool2\_relu** | **(Activation) (None, 56, 56, 256)** | **0** | **['pool2\_bn[0][0]']** |  | | **pool2\_conv** | **(Conv2D) (None, 56, 56, 128)** | **32768** | **['pool2\_relu[0][0]']** |  | | **pool2\_pool** | **(AveragePooling2D) (None, 28, 28, 128)** | **0** | **['pool2\_conv[0][0]']** |  | | **conv3\_block1\_0\_bn** | **(BatchNormal (None, 28, 28, 128)** | **512** | **['pool2\_pool[0][0]']** |  | | **conv3\_block1\_0\_relu** | **(Activatio (None, 28, 28, 128)** | **0** | **['conv3\_block1\_0\_bn[0][0]']** |  | | **conv3\_block1\_1\_conv** | **(Conv2D) (None, 28, 28, 128)** | **16384** | **['conv3\_block1\_0\_relu[0][0]']** |  | | **conv3\_block1\_1\_bn** | **(BatchNormal (None, 28, 28, 128)** | **512** | **['conv3\_block1\_1\_conv[0][0]']** |  | | **conv3\_block1\_1\_relu** | **(Activatio (None, 28, 28, 128)** | **0** | **['conv3\_block1\_1\_bn[0][0]']** |  | | **conv3\_block1\_2\_conv** | **(Conv2D) (None, 28, 28, 32)** | **36864** | **['conv3\_block1\_1\_relu[0][0]']** |  | | **conv3\_block1\_concat** | **(Concatena (None, 28, 28, 160)** | **0** | **['pool2\_pool[0][0]',** |  | | **conv3\_block2\_0\_bn** | **(BatchNormal (None, 28, 28, 160)** | **640** | **['conv3\_block1\_concat[0][0]']** |  | | **conv3\_block2\_0\_relu** | **(Activatio (None, 28, 28, 160)** | **0** | **['conv3\_block2\_0\_bn[0][0]']** |  | | **conv3\_block2\_1\_conv** | **(Conv2D) (None, 28, 28, 128)** | **20480** | **['conv3\_block2\_0\_relu[0][0]']** |  | | **conv3\_block2\_1\_bn** | **(BatchNormal (None, 28, 28, 128)** | **512** | **['conv3\_block2\_1\_conv[0][0]']** |  | | **conv3\_block2\_1\_relu** | **(Activatio (None, 28, 28, 128)** | **0** | **['conv3\_block2\_1\_bn[0][0]']** |  | | **conv3\_block2\_2\_conv** | **(Conv2D) (None, 28, 28, 32)** | **36864** | **['conv3\_block2\_1\_relu[0][0]']** |  | | **conv3\_block2\_concat** | **(Concatena (None, 28, 28, 192)** | **0** | **['conv3\_block1\_concat[0][0]',** |  | | **conv3\_block3\_0\_bn** | **(BatchNormal (None, 28, 28, 192)** | **768** | **['conv3\_block2\_concat[0][0]']** |  | | **conv3\_block3\_0\_relu** | **(Activatio (None, 28, 28, 192)** | **0** | **['conv3\_block3\_0\_bn[0][0]']** |  | | **conv3\_block3\_1\_conv** | **(Conv2D) (None, 28, 28, 128)** | **24576** | **['conv3\_block3\_0\_relu[0][0]']** |  | | **conv3\_block3\_1\_bn** | **(BatchNormal (None, 28, 28, 128)** | **512** | **['conv3\_block3\_1\_conv[0][0]']** |  | | **conv3\_block3\_1\_relu** | **(Activatio (None, 28, 28, 128)** | **0** | **['conv3\_block3\_1\_bn[0][0]']** |  | | **conv3\_block3\_2\_conv** | **(Conv2D) (None, 28, 28, 32)** | **36864** | **['conv3\_block3\_1\_relu[0][0]']** |  | | **conv3\_block3\_concat** | **(Concatena (None, 28, 28, 224)** | **0** | **['conv3\_block2\_concat[0][0]',** |  | | **conv3\_block4\_0\_bn** | **(BatchNormal (None, 28, 28, 224)** | **896** | **['conv3\_block3\_concat[0][0]']** |  | | **conv3\_block4\_0\_relu** | **(Activatio (None, 28, 28, 224)** | **0** | **['conv3\_block4\_0\_bn[0][0]']** |  | | **conv3\_block4\_1\_conv** | **(Conv2D) (None, 28, 28, 128)** | **28672** | **['conv3\_block4\_0\_relu[0][0]']** |  | | **conv3\_block4\_1\_bn** | **(BatchNormal (None, 28, 28, 128)** | **512** | **['conv3\_block4\_1\_conv[0][0]']** |  | | **conv3\_block4\_1\_relu** | **(Activatio (None, 28, 28, 128)** | **0** | **['conv3\_block4\_1\_bn[0][0]']** |  | | **conv3\_block4\_2\_conv** | **(Conv2D) (None, 28, 28, 32)** | **36864** | **['conv3\_block4\_1\_relu[0][0]']** |  | | **conv3\_block4\_concat** | **(Concatena (None, 28, 28, 256)** | **0** | **['conv3\_block3\_concat[0][0]',** |  | | **conv3\_block5\_0\_bn** | **(BatchNormal (None, 28, 28, 256)** | **1024** | **['conv3\_block4\_concat[0][0]']** |  | | **conv3\_block5\_0\_relu** | **(Activatio (None, 28, 28, 256)** | **0** | **['conv3\_block5\_0\_bn[0][0]']** |  | | **conv3\_block5\_1\_conv** | **(Conv2D) (None, 28, 28, 128)** | **32768** | **['conv3\_block5\_0\_relu[0][0]']** |  | | **conv3\_block5\_1\_bn** | **(BatchNormal (None, 28, 28, 128)** | **512** | **['conv3\_block5\_1\_conv[0][0]']** |  | | **conv3\_block5\_1\_relu** | **(Activatio (None, 28, 28, 128)** | **0** | **['conv3\_block5\_1\_bn[0][0]']** |  | | **conv3\_block5\_2\_conv** | **(Conv2D) (None, 28, 28, 32)** | **36864** | **['conv3\_block5\_1\_relu[0][0]']** |  | | **conv3\_block5\_concat** | **(Concatena (None, 28, 28, 288)** | **0** | **['conv3\_block4\_concat[0][0]',** |  | | **conv3\_block6\_0\_bn** | **(BatchNormal (None, 28, 28, 288)** | **1152** | **['conv3\_block5\_concat[0][0]']** |  | | **conv3\_block6\_0\_relu** | **(Activatio (None, 28, 28, 288)** | **0** | **['conv3\_block6\_0\_bn[0][0]']** |  | | **conv3\_block6\_1\_conv** | **(Conv2D) (None, 28, 28, 128)** | **36864** | **['conv3\_block6\_0\_relu[0][0]']** |  | | **conv3\_block6\_1\_bn** | **(BatchNormal (None, 28, 28, 128)** | **512** | **['conv3\_block6\_1\_conv[0][0]']** |  | | **conv3\_block6\_1\_relu** | **(Activatio (None, 28, 28, 128)** | **0** | **['conv3\_block6\_1\_bn[0][0]']** |  | | **conv3\_block6\_2\_conv** | **(Conv2D) (None, 28, 28, 32)** | **36864** | **['conv3\_block6\_1\_relu[0][0]']** |  | | **conv3\_block6\_concat** | **(Concatena (None, 28, 28, 320)** | **0** | **['conv3\_block5\_concat[0][0]',** |  | | **conv3\_block7\_0\_bn** | **(BatchNormal (None, 28, 28, 320)** | **1280** | **['conv3\_block6\_concat[0][0]']** |  | | **conv3\_block7\_0\_relu** | **(Activatio (None, 28, 28, 320)** | **0** | **['conv3\_block7\_0\_bn[0][0]']** |  | | **conv3\_block7\_1\_conv** | **(Conv2D) (None, 28, 28, 128)** | **40960** | **['conv3\_block7\_0\_relu[0][0]']** |  | | **conv3\_block7\_1\_bn** | **(BatchNormal (None, 28, 28, 128)** | **512** | **['conv3\_block7\_1\_conv[0][0]']** |  | | **conv3\_block7\_1\_relu** | **(Activatio (None, 28, 28, 128)** | **0** | **['conv3\_block7\_1\_bn[0][0]']** |  | | **conv3\_block7\_2\_conv** | **(Conv2D) (None, 28, 28, 32)** | **36864** | **['conv3\_block7\_1\_relu[0][0]']** |  | | **conv3\_block7\_concat** | **(Concatena (None, 28, 28, 352)** | **0** | **['conv3\_block6\_concat[0][0]',** |  | | **conv3\_block8\_0\_bn** | **(BatchNormal (None, 28, 28, 352)** | **1408** | **['conv3\_block7\_concat[0][0]']** |  | | **conv3\_block8\_0\_relu** | **(Activatio (None, 28, 28, 352)** | **0** | **['conv3\_block8\_0\_bn[0][0]']** |  | | **conv3\_block8\_1\_conv** | **(Conv2D) (None, 28, 28, 128)** | **45056** | **['conv3\_block8\_0\_relu[0][0]']** |  | | **conv3\_block8\_1\_bn** | **(BatchNormal (None, 28, 28, 128)** | **512** | **['conv3\_block8\_1\_conv[0][0]']** |  | | **conv3\_block8\_1\_relu** | **(Activatio (None, 28, 28, 128)** | **0** | **['conv3\_block8\_1\_bn[0][0]']** |  | | **conv3\_block8\_2\_conv** | **(Conv2D) (None, 28, 28, 32)** | **36864** | **['conv3\_block8\_1\_relu[0][0]']** |  | | **conv3\_block8\_concat** | **(Concatena (None, 28, 28, 384)** | **0** | **['conv3\_block7\_concat[0][0]',** |  | | **conv3\_block9\_0\_bn** | **(BatchNormal (None, 28, 28, 384)** | **1536** | **['conv3\_block8\_concat[0][0]']** |  | | **conv3\_block9\_0\_relu** | **(Activatio (None, 28, 28, 384)** | **0** | **['conv3\_block9\_0\_bn[0][0]']** |  | | **conv3\_block9\_1\_conv** | **(Conv2D) (None, 28, 28, 128)** | **49152** | **['conv3\_block9\_0\_relu[0][0]']** |  | | **conv3\_block9\_1\_bn** | **(BatchNormal (None, 28, 28, 128)** | **512** | **['conv3\_block9\_1\_conv[0][0]']** |  | | **conv3\_block9\_1\_relu** | **(Activatio (None, 28, 28, 128)** | **0** | **['conv3\_block9\_1\_bn[0][0]']** |  | | **conv3\_block9\_2\_conv** | **(Conv2D) (None, 28, 28, 32)** | **36864** | **['conv3\_block9\_1\_relu[0][0]']** |  | | **conv3\_block9\_concat** | **(Concatena (None, 28, 28, 416)** | **0** | **['conv3\_block8\_concat[0][0]',** |  | | **conv3\_block10\_0\_bn** | **(BatchNorma (None, 28, 28, 416)** | **1664** | **['conv3\_block9\_concat[0][0]']** |  | | **conv3\_block10\_0\_relu** | **(Activati (None, 28, 28, 416)** | **0** | **['conv3\_block10\_0\_bn[0][0]']** |  | | **conv3\_block10\_1\_conv** | **(Conv2D) (None, 28, 28, 128)** | **53248** | **['conv3\_block10\_0\_relu[0][0]']** |  | | **conv3\_block10\_1\_bn** | **(BatchNorma (None, 28, 28, 128)** | **512** | **['conv3\_block10\_1\_conv[0][0]']** |  | | **conv3\_block10\_1\_relu** | **(Activati (None, 28, 28, 128)** | **0** | **['conv3\_block10\_1\_bn[0][0]']** |  | | **conv3\_block10\_2\_conv** | **(Conv2D) (None, 28, 28, 32)** | **36864** | **['conv3\_block10\_1\_relu[0][0]']** |  | | **conv3\_block10\_concat** | **(Concaten (None, 28, 28, 448)** | **0** | **['conv3\_block9\_concat[0][0]',** |  | | **conv3\_block11\_0\_bn** | **(BatchNorma (None, 28, 28, 448)** | **1792** | **['conv3\_block10\_concat[0][0]']** |  | | **conv3\_block11\_0\_relu** | **(Activati (None, 28, 28, 448)** | **0** | **['conv3\_block11\_0\_bn[0][0]']** |  | | **conv3\_block11\_1\_conv** | **(Conv2D) (None, 28, 28, 128)** | **57344** | **['conv3\_block11\_0\_relu[0][0]']** |  | | **conv3\_block11\_1\_bn** | **(BatchNorma (None, 28, 28, 128)** | **512** | **['conv3\_block11\_1\_conv[0][0]']** |  | | **conv3\_block11\_1\_relu** | **(Activati (None, 28, 28, 128)** | **0** | **['conv3\_block11\_1\_bn[0][0]']** |  | | **conv3\_block11\_2\_conv** | **(Conv2D) (None, 28, 28, 32)** | **36864** | **['conv3\_block11\_1\_relu[0][0]']** |  | | **conv3\_block11\_concat** | **(Concaten (None, 28, 28, 480)** | **0** | **['conv3\_block10\_concat[0][0]',** |  | | **conv3\_block12\_0\_bn** | **(BatchNorma (None, 28, 28, 480)** | **1920** | **['conv3\_block11\_concat[0][0]']** |  | | **conv3\_block12\_0\_relu** | **(Activati (None, 28, 28, 480)** | **0** | **['conv3\_block12\_0\_bn[0][0]']** |  | | **conv3\_block12\_1\_conv** | **(Conv2D) (None, 28, 28, 128)** | **61440** | **['conv3\_block12\_0\_relu[0][0]']** |  | | **conv3\_block12\_1\_bn** | **(BatchNorma (None, 28, 28, 128)** | **512** | **['conv3\_block12\_1\_conv[0][0]']** |  | | **conv3\_block12\_1\_relu** | **(Activati (None, 28, 28, 128)** | **0** | **['conv3\_block12\_1\_bn[0][0]']** |  | | **conv3\_block12\_2\_conv** | **(Conv2D) (None, 28, 28, 32)** | **36864** | **['conv3\_block12\_1\_relu[0][0]']** |  | | **conv3\_block12\_concat** | **(Concaten (None, 28, 28, 512)** | **0** | **['conv3\_block11\_concat[0][0]',** |  | | **pool3\_bn** | **(BatchNormalization) (None, 28, 28, 512)** | **2048** | **['conv3\_block12\_concat[0][0]']** |  | | **pool3\_relu** | **(Activation) (None, 28, 28, 512)** | **0** | **['pool3\_bn[0][0]']** |  | | **pool3\_conv** | **(Conv2D) (None, 28, 28, 256)** | **131072** | **['pool3\_relu[0][0]']** |  | | **pool3\_pool** | **(AveragePooling2D) (None, 14, 14, 256)** | **0** | **['pool3\_conv[0][0]']** |  | | **conv4\_block1\_0\_bn** | **(BatchNormal (None, 14, 14, 256)** | **1024** | **['pool3\_pool[0][0]']** |  | | **conv4\_block1\_0\_relu** | **(Activatio (None, 14, 14, 256)** | **0** | **['conv4\_block1\_0\_bn[0][0]']** |  | | **conv4\_block1\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **32768** | **['conv4\_block1\_0\_relu[0][0]']** |  | | **conv4\_block1\_1\_bn** | **(BatchNormal (None, 14, 14, 128)** | **512** | **['conv4\_block1\_1\_conv[0][0]']** |  | | **conv4\_block1\_1\_relu** | **(Activatio (None, 14, 14, 128)** | **0** | **['conv4\_block1\_1\_bn[0][0]']** |  | | **conv4\_block1\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block1\_1\_relu[0][0]']** |  | | **conv4\_block1\_concat** | **(Concatena (None, 14, 14, 288)** | **0** | **['pool3\_pool[0][0]',** |  | | **conv4\_block2\_0\_bn** | **(BatchNormal (None, 14, 14, 288)** | **1152** | **['conv4\_block1\_concat[0][0]']** |  | | **conv4\_block2\_0\_relu** | **(Activatio (None, 14, 14, 288)** | **0** | **['conv4\_block2\_0\_bn[0][0]']** |  | | **conv4\_block2\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **36864** | **['conv4\_block2\_0\_relu[0][0]']** |  | | **conv4\_block2\_1\_bn** | **(BatchNormal (None, 14, 14, 128)** | **512** | **['conv4\_block2\_1\_conv[0][0]']** |  | | **conv4\_block2\_1\_relu** | **(Activatio (None, 14, 14, 128)** | **0** | **['conv4\_block2\_1\_bn[0][0]']** |  | | **conv4\_block2\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block2\_1\_relu[0][0]']** |  | | **conv4\_block2\_concat** | **(Concatena (None, 14, 14, 320)** | **0** | **['conv4\_block1\_concat[0][0]',** |  | | **conv4\_block3\_0\_bn** | **(BatchNormal (None, 14, 14, 320)** | **1280** | **['conv4\_block2\_concat[0][0]']** |  | | **conv4\_block3\_0\_relu** | **(Activatio (None, 14, 14, 320)** | **0** | **['conv4\_block3\_0\_bn[0][0]']** |  | | **conv4\_block3\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **40960** | **['conv4\_block3\_0\_relu[0][0]']** |  | | **conv4\_block3\_1\_bn** | **(BatchNormal (None, 14, 14, 128)** | **512** | **['conv4\_block3\_1\_conv[0][0]']** |  | | **conv4\_block3\_1\_relu** | **(Activatio (None, 14, 14, 128)** | **0** | **['conv4\_block3\_1\_bn[0][0]']** |  | | **conv4\_block3\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block3\_1\_relu[0][0]']** |  | | **conv4\_block3\_concat** | **(Concatena (None, 14, 14, 352)** | **0** | **['conv4\_block2\_concat[0][0]',** |  | | **conv4\_block4\_0\_bn** | **(BatchNormal (None, 14, 14, 352)** | **1408** | **['conv4\_block3\_concat[0][0]']** |  | | **conv4\_block4\_0\_relu** | **(Activatio (None, 14, 14, 352)** | **0** | **['conv4\_block4\_0\_bn[0][0]']** |  | | **conv4\_block4\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **45056** | **['conv4\_block4\_0\_relu[0][0]']** |  | | **conv4\_block4\_1\_bn** | **(BatchNormal (None, 14, 14, 128)** | **512** | **['conv4\_block4\_1\_conv[0][0]']** |  | | **conv4\_block4\_1\_relu** | **(Activatio (None, 14, 14, 128)** | **0** | **['conv4\_block4\_1\_bn[0][0]']** |  | | **conv4\_block4\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block4\_1\_relu[0][0]']** |  | | **conv4\_block4\_concat** | **(Concatena (None, 14, 14, 384)** | **0** | **['conv4\_block3\_concat[0][0]',** |  | | **conv4\_block5\_0\_bn** | **(BatchNormal (None, 14, 14, 384)** | **1536** | **['conv4\_block4\_concat[0][0]']** |  | | **conv4\_block5\_0\_relu** | **(Activatio (None, 14, 14, 384)** | **0** | **['conv4\_block5\_0\_bn[0][0]']** |  | | **conv4\_block5\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **49152** | **['conv4\_block5\_0\_relu[0][0]']** |  | | **conv4\_block5\_1\_bn** | **(BatchNormal (None, 14, 14, 128)** | **512** | **['conv4\_block5\_1\_conv[0][0]']** |  | | **conv4\_block5\_1\_relu** | **(Activatio (None, 14, 14, 128)** | **0** | **['conv4\_block5\_1\_bn[0][0]']** |  | | **conv4\_block5\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block5\_1\_relu[0][0]']** |  | | **conv4\_block5\_concat** | **(Concatena (None, 14, 14, 416)** | **0** | **['conv4\_block4\_concat[0][0]',** |  | | **conv4\_block6\_0\_bn** | **(BatchNormal (None, 14, 14, 416)** | **1664** | **['conv4\_block5\_concat[0][0]']** |  | | **conv4\_block6\_0\_relu** | **(Activatio (None, 14, 14, 416)** | **0** | **['conv4\_block6\_0\_bn[0][0]']** |  | | **conv4\_block6\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **53248** | **['conv4\_block6\_0\_relu[0][0]']** |  | | **conv4\_block6\_1\_bn** | **(BatchNormal (None, 14, 14, 128)** | **512** | **['conv4\_block6\_1\_conv[0][0]']** |  | | **conv4\_block6\_1\_relu** | **(Activatio (None, 14, 14, 128)** | **0** | **['conv4\_block6\_1\_bn[0][0]']** |  | | **conv4\_block6\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block6\_1\_relu[0][0]']** |  | | **conv4\_block6\_concat** | **(Concatena (None, 14, 14, 448)** | **0** | **['conv4\_block5\_concat[0][0]',** |  | | **conv4\_block7\_0\_bn** | **(BatchNormal (None, 14, 14, 448)** | **1792** | **['conv4\_block6\_concat[0][0]']** |  | | **conv4\_block7\_0\_relu** | **(Activatio (None, 14, 14, 448)** | **0** | **['conv4\_block7\_0\_bn[0][0]']** |  | | **conv4\_block7\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **57344** | **['conv4\_block7\_0\_relu[0][0]']** |  | | **conv4\_block7\_1\_bn** | **(BatchNormal (None, 14, 14, 128)** | **512** | **['conv4\_block7\_1\_conv[0][0]']** |  | | **conv4\_block7\_1\_relu** | **(Activatio (None, 14, 14, 128)** | **0** | **['conv4\_block7\_1\_bn[0][0]']** |  | | **conv4\_block7\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block7\_1\_relu[0][0]']** |  | | **conv4\_block7\_concat** | **(Concatena (None, 14, 14, 480)** | **0** | **['conv4\_block6\_concat[0][0]',** |  | | **conv4\_block8\_0\_bn** | **(BatchNormal (None, 14, 14, 480)** | **1920** | **['conv4\_block7\_concat[0][0]']** |  | | **conv4\_block8\_0\_relu** | **(Activatio (None, 14, 14, 480)** | **0** | **['conv4\_block8\_0\_bn[0][0]']** |  | | **conv4\_block8\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **61440** | **['conv4\_block8\_0\_relu[0][0]']** |  | | **conv4\_block8\_1\_bn** | **(BatchNormal (None, 14, 14, 128)** | **512** | **['conv4\_block8\_1\_conv[0][0]']** |  | | **conv4\_block8\_1\_relu** | **(Activatio (None, 14, 14, 128)** | **0** | **['conv4\_block8\_1\_bn[0][0]']** |  | | **conv4\_block8\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block8\_1\_relu[0][0]']** |  | | **conv4\_block8\_concat** | **(Concatena (None, 14, 14, 512)** | **0** | **['conv4\_block7\_concat[0][0]',** |  | | **conv4\_block9\_0\_bn** | **(BatchNormal (None, 14, 14, 512)** | **2048** | **['conv4\_block8\_concat[0][0]']** |  | | **conv4\_block9\_0\_relu** | **(Activatio (None, 14, 14, 512)** | **0** | **['conv4\_block9\_0\_bn[0][0]']** |  | | **conv4\_block9\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **65536** | **['conv4\_block9\_0\_relu[0][0]']** |  | | **conv4\_block9\_1\_bn** | **(BatchNormal (None, 14, 14, 128)** | **512** | **['conv4\_block9\_1\_conv[0][0]']** |  | | **conv4\_block9\_1\_relu** | **(Activatio (None, 14, 14, 128)** | **0** | **['conv4\_block9\_1\_bn[0][0]']** |  | | **conv4\_block9\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block9\_1\_relu[0][0]']** |  | | **conv4\_block9\_concat** | **(Concatena (None, 14, 14, 544)** | **0** | **['conv4\_block8\_concat[0][0]',** |  | | **conv4\_block10\_0\_bn** | **(BatchNorma (None, 14, 14, 544)** | **2176** | **['conv4\_block9\_concat[0][0]']** |  | | **conv4\_block10\_0\_relu** | **(Activati (None, 14, 14, 544)** | **0** | **['conv4\_block10\_0\_bn[0][0]']** |  | | **conv4\_block10\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **69632** | **['conv4\_block10\_0\_relu[0][0]']** |  | | **conv4\_block10\_1\_bn** | **(BatchNorma (None, 14, 14, 128)** | **512** | **['conv4\_block10\_1\_conv[0][0]']** |  | | **conv4\_block10\_1\_relu** | **(Activati (None, 14, 14, 128)** | **0** | **['conv4\_block10\_1\_bn[0][0]']** |  | | **conv4\_block10\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block10\_1\_relu[0][0]']** |  | | **conv4\_block10\_concat** | **(Concaten (None, 14, 14, 576)** | **0** | **['conv4\_block9\_concat[0][0]',** |  | | **conv4\_block11\_0\_bn** | **(BatchNorma (None, 14, 14, 576)** | **2304** | **['conv4\_block10\_concat[0][0]']** |  | | **conv4\_block11\_0\_relu** | **(Activati (None, 14, 14, 576)** | **0** | **['conv4\_block11\_0\_bn[0][0]']** |  | | **conv4\_block11\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **73728** | **['conv4\_block11\_0\_relu[0][0]']** |  | | **conv4\_block11\_1\_bn** | **(BatchNorma (None, 14, 14, 128)** | **512** | **['conv4\_block11\_1\_conv[0][0]']** |  | | **conv4\_block11\_1\_relu** | **(Activati (None, 14, 14, 128)** | **0** | **['conv4\_block11\_1\_bn[0][0]']** |  | | **conv4\_block11\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block11\_1\_relu[0][0]']** |  | | **conv4\_block11\_concat** | **(Concaten (None, 14, 14, 608)** | **0** | **['conv4\_block10\_concat[0][0]',** |  | | **conv4\_block12\_0\_bn** | **(BatchNorma (None, 14, 14, 608)** | **2432** | **['conv4\_block11\_concat[0][0]']** |  | | **conv4\_block12\_0\_relu** | **(Activati (None, 14, 14, 608)** | **0** | **['conv4\_block12\_0\_bn[0][0]']** |  | | **conv4\_block12\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **77824** | **['conv4\_block12\_0\_relu[0][0]']** |  | | **conv4\_block12\_1\_bn** | **(BatchNorma (None, 14, 14, 128)** | **512** | **['conv4\_block12\_1\_conv[0][0]']** |  | | **conv4\_block12\_1\_relu** | **(Activati (None, 14, 14, 128)** | **0** | **['conv4\_block12\_1\_bn[0][0]']** |  | | **conv4\_block12\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block12\_1\_relu[0][0]']** |  | | **conv4\_block12\_concat** | **(Concaten (None, 14, 14, 640)** | **0** | **['conv4\_block11\_concat[0][0]',** |  | | **conv4\_block13\_0\_bn** | **(BatchNorma (None, 14, 14, 640)** | **2560** | **['conv4\_block12\_concat[0][0]']** |  | | **conv4\_block13\_0\_relu** | **(Activati (None, 14, 14, 640)** | **0** | **['conv4\_block13\_0\_bn[0][0]']** |  | | **conv4\_block13\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **81920** | **['conv4\_block13\_0\_relu[0][0]']** |  | | **conv4\_block13\_1\_bn** | **(BatchNorma (None, 14, 14, 128)** | **512** | **['conv4\_block13\_1\_conv[0][0]']** |  | | **conv4\_block13\_1\_relu** | **(Activati (None, 14, 14, 128)** | **0** | **['conv4\_block13\_1\_bn[0][0]']** |  | | **conv4\_block13\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block13\_1\_relu[0][0]']** |  | | **conv4\_block13\_concat** | **(Concaten (None, 14, 14, 672)** | **0** | **['conv4\_block12\_concat[0][0]',** |  | | **conv4\_block14\_0\_bn** | **(BatchNorma (None, 14, 14, 672)** | **2688** | **['conv4\_block13\_concat[0][0]']** |  | | **conv4\_block14\_0\_relu** | **(Activati (None, 14, 14, 672)** | **0** | **['conv4\_block14\_0\_bn[0][0]']** |  | | **conv4\_block14\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **86016** | **['conv4\_block14\_0\_relu[0][0]']** |  | | **conv4\_block14\_1\_bn** | **(BatchNorma (None, 14, 14, 128)** | **512** | **['conv4\_block14\_1\_conv[0][0]']** |  | | **conv4\_block14\_1\_relu** | **(Activati (None, 14, 14, 128)** | **0** | **['conv4\_block14\_1\_bn[0][0]']** |  | | **conv4\_block14\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block14\_1\_relu[0][0]']** |  | | **conv4\_block14\_concat** | **(Concaten (None, 14, 14, 704)** | **0** | **['conv4\_block13\_concat[0][0]',** |  | | **conv4\_block15\_0\_bn** | **(BatchNorma (None, 14, 14, 704)** | **2816** | **['conv4\_block14\_concat[0][0]']** |  | | **conv4\_block15\_0\_relu** | **(Activati (None, 14, 14, 704)** | **0** | **['conv4\_block15\_0\_bn[0][0]']** |  | | **conv4\_block15\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **90112** | **['conv4\_block15\_0\_relu[0][0]']** |  | | **conv4\_block15\_1\_bn** | **(BatchNorma (None, 14, 14, 128)** | **512** | **['conv4\_block15\_1\_conv[0][0]']** |  | | **conv4\_block15\_1\_relu** | **(Activati (None, 14, 14, 128)** | **0** | **['conv4\_block15\_1\_bn[0][0]']** |  | | **conv4\_block15\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block15\_1\_relu[0][0]']** |  | | **conv4\_block15\_concat** | **(Concaten (None, 14, 14, 736)** | **0** | **['conv4\_block14\_concat[0][0]',** |  | | **conv4\_block16\_0\_bn** | **(BatchNorma (None, 14, 14, 736)** | **2944** | **['conv4\_block15\_concat[0][0]']** |  | | **conv4\_block16\_0\_relu** | **(Activati (None, 14, 14, 736)** | **0** | **['conv4\_block16\_0\_bn[0][0]']** |  | | **conv4\_block16\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **94208** | **['conv4\_block16\_0\_relu[0][0]']** |  | | **conv4\_block16\_1\_bn** | **(BatchNorma (None, 14, 14, 128)** | **512** | **['conv4\_block16\_1\_conv[0][0]']** |  | | **conv4\_block16\_1\_relu** | **(Activati (None, 14, 14, 128)** | **0** | **['conv4\_block16\_1\_bn[0][0]']** |  | | **conv4\_block16\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block16\_1\_relu[0][0]']** |  | | **conv4\_block16\_concat** | **(Concaten (None, 14, 14, 768)** | **0** | **['conv4\_block15\_concat[0][0]',** |  | | **conv4\_block17\_0\_bn** | **(BatchNorma (None, 14, 14, 768)** | **3072** | **['conv4\_block16\_concat[0][0]']** |  | | **conv4\_block17\_0\_relu** | **(Activati (None, 14, 14, 768)** | **0** | **['conv4\_block17\_0\_bn[0][0]']** |  | | **conv4\_block17\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **98304** | **['conv4\_block17\_0\_relu[0][0]']** |  | | **conv4\_block17\_1\_bn** | **(BatchNorma (None, 14, 14, 128)** | **512** | **['conv4\_block17\_1\_conv[0][0]']** |  | | **conv4\_block17\_1\_relu** | **(Activati (None, 14, 14, 128)** | **0** | **['conv4\_block17\_1\_bn[0][0]']** |  | | **conv4\_block17\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block17\_1\_relu[0][0]']** |  | | **conv4\_block17\_concat** | **(Concaten (None, 14, 14, 800)** | **0** | **['conv4\_block16\_concat[0][0]',** |  | | **conv4\_block18\_0\_bn** | **(BatchNorma (None, 14, 14, 800)** | **3200** | **['conv4\_block17\_concat[0][0]']** |  | | **conv4\_block18\_0\_relu** | **(Activati (None, 14, 14, 800)** | **0** | **['conv4\_block18\_0\_bn[0][0]']** |  | | **conv4\_block18\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **102400** | **['conv4\_block18\_0\_relu[0][0]']** |  | | **conv4\_block18\_1\_bn** | **(BatchNorma (None, 14, 14, 128)** | **512** | **['conv4\_block18\_1\_conv[0][0]']** |  | | **conv4\_block18\_1\_relu** | **(Activati (None, 14, 14, 128)** | **0** | **['conv4\_block18\_1\_bn[0][0]']** |  | | **conv4\_block18\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block18\_1\_relu[0][0]']** |  | | **conv4\_block18\_concat** | **(Concaten (None, 14, 14, 832)** | **0** | **['conv4\_block17\_concat[0][0]',** |  | | **conv4\_block19\_0\_bn** | **(BatchNorma (None, 14, 14, 832)** | **3328** | **['conv4\_block18\_concat[0][0]']** |  | | **conv4\_block19\_0\_relu** | **(Activati (None, 14, 14, 832)** | **0** | **['conv4\_block19\_0\_bn[0][0]']** |  | | **conv4\_block19\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **106496** | **['conv4\_block19\_0\_relu[0][0]']** |  | | **conv4\_block19\_1\_bn** | **(BatchNorma (None, 14, 14, 128)** | **512** | **['conv4\_block19\_1\_conv[0][0]']** |  | | **conv4\_block19\_1\_relu** | **(Activati (None, 14, 14, 128)** | **0** | **['conv4\_block19\_1\_bn[0][0]']** |  | | **conv4\_block19\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block19\_1\_relu[0][0]']** |  | | **conv4\_block19\_concat** | **(Concaten (None, 14, 14, 864)** | **0** | **['conv4\_block18\_concat[0][0]',** |  | | **conv4\_block20\_0\_bn** | **(BatchNorma (None, 14, 14, 864)** | **3456** | **['conv4\_block19\_concat[0][0]']** |  | | **conv4\_block20\_0\_relu** | **(Activati (None, 14, 14, 864)** | **0** | **['conv4\_block20\_0\_bn[0][0]']** |  | | **conv4\_block20\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **110592** | **['conv4\_block20\_0\_relu[0][0]']** |  | | **conv4\_block20\_1\_bn** | **(BatchNorma (None, 14, 14, 128)** | **512** | **['conv4\_block20\_1\_conv[0][0]']** |  | | **conv4\_block20\_1\_relu** | **(Activati (None, 14, 14, 128)** | **0** | **['conv4\_block20\_1\_bn[0][0]']** |  | | **conv4\_block20\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block20\_1\_relu[0][0]']** |  | | **conv4\_block20\_concat** | **(Concaten (None, 14, 14, 896)** | **0** | **['conv4\_block19\_concat[0][0]',** |  | | **conv4\_block21\_0\_bn** | **(BatchNorma (None, 14, 14, 896)** | **3584** | **['conv4\_block20\_concat[0][0]']** |  | | **conv4\_block21\_0\_relu** | **(Activati (None, 14, 14, 896)** | **0** | **['conv4\_block21\_0\_bn[0][0]']** |  | | **conv4\_block21\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **114688** | **['conv4\_block21\_0\_relu[0][0]']** |  | | **conv4\_block21\_1\_bn** | **(BatchNorma (None, 14, 14, 128)** | **512** | **['conv4\_block21\_1\_conv[0][0]']** |  | | **conv4\_block21\_1\_relu** | **(Activati (None, 14, 14, 128)** | **0** | **['conv4\_block21\_1\_bn[0][0]']** |  | | **conv4\_block21\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block21\_1\_relu[0][0]']** |  | | **conv4\_block21\_concat** | **(Concaten (None, 14, 14, 928)** | **0** | **['conv4\_block20\_concat[0][0]',** |  | | **conv4\_block22\_0\_bn** | **(BatchNorma (None, 14, 14, 928)** | **3712** | **['conv4\_block21\_concat[0][0]']** |  | | **conv4\_block22\_0\_relu** | **(Activati (None, 14, 14, 928)** | **0** | **['conv4\_block22\_0\_bn[0][0]']** |  | | **conv4\_block22\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **118784** | **['conv4\_block22\_0\_relu[0][0]']** |  | | **conv4\_block22\_1\_bn** | **(BatchNorma (None, 14, 14, 128)** | **512** | **['conv4\_block22\_1\_conv[0][0]']** |  | | **conv4\_block22\_1\_relu** | **(Activati (None, 14, 14, 128)** | **0** | **['conv4\_block22\_1\_bn[0][0]']** |  | | **conv4\_block22\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block22\_1\_relu[0][0]']** |  | | **conv4\_block22\_concat** | **(Concaten (None, 14, 14, 960)** | **0** | **['conv4\_block21\_concat[0][0]',** |  | | **conv4\_block23\_0\_bn** | **(BatchNorma (None, 14, 14, 960)** | **3840** | **['conv4\_block22\_concat[0][0]']** |  | | **conv4\_block23\_0\_relu** | **(Activati (None, 14, 14, 960)** | **0** | **['conv4\_block23\_0\_bn[0][0]']** |  | | **conv4\_block23\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **122880** | **['conv4\_block23\_0\_relu[0][0]']** |  | | **conv4\_block23\_1\_bn** | **(BatchNorma (None, 14, 14, 128)** | **512** | **['conv4\_block23\_1\_conv[0][0]']** |  | | **conv4\_block23\_1\_relu** | **(Activati (None, 14, 14, 128)** | **0** | **['conv4\_block23\_1\_bn[0][0]']** |  | | **conv4\_block23\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block23\_1\_relu[0][0]']** |  | | **conv4\_block23\_concat** | **(Concaten (None, 14, 14, 992)** | **0** | **['conv4\_block22\_concat[0][0]',** |  | | **conv4\_block24\_0\_bn** | **(BatchNorma (None, 14, 14, 992)** | **3968** | **['conv4\_block23\_concat[0][0]']** |  | | **conv4\_block24\_0\_relu** | **(Activati (None, 14, 14, 992)** | **0** | **['conv4\_block24\_0\_bn[0][0]']** |  | | **conv4\_block24\_1\_conv** | **(Conv2D) (None, 14, 14, 128)** | **126976** | **['conv4\_block24\_0\_relu[0][0]']** |  | | **conv4\_block24\_1\_bn** | **(BatchNorma (None, 14, 14, 128)** | **512** | **['conv4\_block24\_1\_conv[0][0]']** |  | | **conv4\_block24\_1\_relu** | **(Activati (None, 14, 14, 128)** | **0** | **['conv4\_block24\_1\_bn[0][0]']** |  | | **conv4\_block24\_2\_conv** | **(Conv2D) (None, 14, 14, 32)** | **36864** | **['conv4\_block24\_1\_relu[0][0]']** |  | | **conv4\_block24\_concat** | **(Concaten (None, 14, 14, 1024** | **0** | **['conv4\_block23\_concat[0][0]',** |  | | **pool4\_bn** | **(BatchNormalization) (None, 14, 14, 1024** | **4096** | **['conv4\_block24\_concat[0][0]']** |  | | **pool4\_relu** | **(Activation) (None, 14, 14, 1024** | **0** | **['pool4\_bn[0][0]']** |  | | **pool4\_conv** | **(Conv2D) (None, 14, 14, 512)** | **524288** | **['pool4\_relu[0][0]']** |  | | **pool4\_pool** | **(AveragePooling2D) (None, 7, 7, 512)** | **0** | **['pool4\_conv[0][0]']** |  | | **conv5\_block1\_0\_bn** | **(BatchNormal (None, 7, 7, 512)** | **2048** | **['pool4\_pool[0][0]']** |  | | **conv5\_block1\_0\_relu** | **(Activatio (None, 7, 7, 512)** | **0** | **['conv5\_block1\_0\_bn[0][0]']** |  | | **conv5\_block1\_1\_conv** | **(Conv2D) (None, 7, 7, 128)** | **65536** | **['conv5\_block1\_0\_relu[0][0]']** |  | | **conv5\_block1\_1\_bn** | **(BatchNormal (None, 7, 7, 128)** | **512** | **['conv5\_block1\_1\_conv[0][0]']** |  | | **conv5\_block1\_1\_relu** | **(Activatio (None, 7, 7, 128)** | **0** | **['conv5\_block1\_1\_bn[0][0]']** |  | | **conv5\_block1\_2\_conv** | **(Conv2D) (None, 7, 7, 32)** | **36864** | **['conv5\_block1\_1\_relu[0][0]']** |  | | **conv5\_block1\_concat** | **(Concatena (None, 7, 7, 544)** | **0** | **['pool4\_pool[0][0]',** |  | | **conv5\_block2\_0\_bn** | **(BatchNormal (None, 7, 7, 544)** | **2176** | **['conv5\_block1\_concat[0][0]']** |  | | **conv5\_block2\_0\_relu** | **(Activatio (None, 7, 7, 544)** | **0** | **['conv5\_block2\_0\_bn[0][0]']** |  | | **conv5\_block2\_1\_conv** | **(Conv2D) (None, 7, 7, 128)** | **69632** | **['conv5\_block2\_0\_relu[0][0]']** |  | | **conv5\_block2\_1\_bn** | **(BatchNormal (None, 7, 7, 128)** | **512** | **['conv5\_block2\_1\_conv[0][0]']** |  | | **conv5\_block2\_1\_relu** | **(Activatio (None, 7, 7, 128)** | **0** | **['conv5\_block2\_1\_bn[0][0]']** |  | | **conv5\_block2\_2\_conv** | **(Conv2D) (None, 7, 7, 32)** | **36864** | **['conv5\_block2\_1\_relu[0][0]']** |  | | **conv5\_block2\_concat** | **(Concatena (None, 7, 7, 576)** | **0** | **['conv5\_block1\_concat[0][0]',** |  | | **conv5\_block3\_0\_bn** | **(BatchNormal (None, 7, 7, 576)** | **2304** | **['conv5\_block2\_concat[0][0]']** |  | | **conv5\_block3\_0\_relu** | **(Activatio (None, 7, 7, 576)** | **0** | **['conv5\_block3\_0\_bn[0][0]']** |  | | **conv5\_block3\_1\_conv** | **(Conv2D) (None, 7, 7, 128)** | **73728** | **['conv5\_block3\_0\_relu[0][0]']** |  | | **conv5\_block3\_1\_bn** | **(BatchNormal (None, 7, 7, 128)** | **512** | **['conv5\_block3\_1\_conv[0][0]']** |  | | **conv5\_block3\_1\_relu** | **(Activatio (None, 7, 7, 128)** | **0** | **['conv5\_block3\_1\_bn[0][0]']** |  | | **conv5\_block3\_2\_conv** | **(Conv2D) (None, 7, 7, 32)** | **36864** | **['conv5\_block3\_1\_relu[0][0]']** |  | | **conv5\_block3\_concat** | **(Concatena (None, 7, 7, 608)** | **0** | **['conv5\_block2\_concat[0][0]',** |  | | **conv5\_block4\_0\_bn** | **(BatchNormal (None, 7, 7, 608)** | **2432** | **['conv5\_block3\_concat[0][0]']** |  | | **conv5\_block4\_0\_relu** | **(Activatio (None, 7, 7, 608)** | **0** | **['conv5\_block4\_0\_bn[0][0]']** |  | | **conv5\_block4\_1\_conv** | **(Conv2D) (None, 7, 7, 128)** | **77824** | **['conv5\_block4\_0\_relu[0][0]']** |  | | **conv5\_block4\_1\_bn** | **(BatchNormal (None, 7, 7, 128)** | **512** | **['conv5\_block4\_1\_conv[0][0]']** |  | | **conv5\_block4\_1\_relu** | **(Activatio (None, 7, 7, 128)** | **0** | **['conv5\_block4\_1\_bn[0][0]']** |  | | **conv5\_block4\_2\_conv** | **(Conv2D) (None, 7, 7, 32)** | **36864** | **['conv5\_block4\_1\_relu[0][0]']** |  | | **conv5\_block4\_concat** | **(Concatena (None, 7, 7, 640)** | **0** | **['conv5\_block3\_concat[0][0]',** |  | | **conv5\_block5\_0\_bn** | **(BatchNormal (None, 7, 7, 640)** | **2560** | **['conv5\_block4\_concat[0][0]']** |  | | **conv5\_block5\_0\_relu** | **(Activatio (None, 7, 7, 640)** | **0** | **['conv5\_block5\_0\_bn[0][0]']** |  | | **conv5\_block5\_1\_conv** | **(Conv2D) (None, 7, 7, 128)** | **81920** | **['conv5\_block5\_0\_relu[0][0]']** |  | | **conv5\_block5\_1\_bn** | **(BatchNormal (None, 7, 7, 128)** | **512** | **['conv5\_block5\_1\_conv[0][0]']** |  | | **conv5\_block5\_1\_relu** | **(Activatio (None, 7, 7, 128)** | **0** | **['conv5\_block5\_1\_bn[0][0]']** |  | | **conv5\_block5\_2\_conv** | **(Conv2D) (None, 7, 7, 32)** | **36864** | **['conv5\_block5\_1\_relu[0][0]']** |  | | **conv5\_block5\_concat** | **(Concatena (None, 7, 7, 672)** | **0** | **['conv5\_block4\_concat[0][0]',** |  | | **conv5\_block6\_0\_bn** | **(BatchNormal (None, 7, 7, 672)** | **2688** | **['conv5\_block5\_concat[0][0]']** |  | | **conv5\_block6\_0\_relu** | **(Activatio (None, 7, 7, 672)** | **0** | **['conv5\_block6\_0\_bn[0][0]']** |  | | **conv5\_block6\_1\_conv** | **(Conv2D) (None, 7, 7, 128)** | **86016** | **['conv5\_block6\_0\_relu[0][0]']** |  | | **conv5\_block6\_1\_bn** | **(BatchNormal (None, 7, 7, 128)** | **512** | **['conv5\_block6\_1\_conv[0][0]']** |  | | **conv5\_block6\_1\_relu** | **(Activatio (None, 7, 7, 128)** | **0** | **['conv5\_block6\_1\_bn[0][0]']** |  | | **conv5\_block6\_2\_conv** | **(Conv2D) (None, 7, 7, 32)** | **36864** | **['conv5\_block6\_1\_relu[0][0]']** |  | | **conv5\_block6\_concat** | **(Concatena (None, 7, 7, 704)** | **0** | **['conv5\_block5\_concat[0][0]',** |  | | **conv5\_block7\_0\_bn** | **(BatchNormal (None, 7, 7, 704)** | **2816** | **['conv5\_block6\_concat[0][0]']** |  | | **conv5\_block7\_0\_relu** | **(Activatio (None, 7, 7, 704)** | **0** | **['conv5\_block7\_0\_bn[0][0]']** |  | | **conv5\_block7\_1\_conv** | **(Conv2D) (None, 7, 7, 128)** | **90112** | **['conv5\_block7\_0\_relu[0][0]']** |  | | **conv5\_block7\_1\_bn** | **(BatchNormal (None, 7, 7, 128)** | **512** | **['conv5\_block7\_1\_conv[0][0]']** |  | | **conv5\_block7\_1\_relu** | **(Activatio (None, 7, 7, 128)** | **0** | **['conv5\_block7\_1\_bn[0][0]']** |  | | **conv5\_block7\_2\_conv** | **(Conv2D) (None, 7, 7, 32)** | **36864** | **['conv5\_block7\_1\_relu[0][0]']** |  | | **conv5\_block7\_concat** | **(Concatena (None, 7, 7, 736)** | **0** | **['conv5\_block6\_concat[0][0]',** |  | | **conv5\_block8\_0\_bn** | **(BatchNormal (None, 7, 7, 736)** | **2944** | **['conv5\_block7\_concat[0][0]']** |  | | **conv5\_block8\_0\_relu** | **(Activatio (None, 7, 7, 736)** | **0** | **['conv5\_block8\_0\_bn[0][0]']** |  | | **conv5\_block8\_1\_conv** | **(Conv2D) (None, 7, 7, 128)** | **94208** | **['conv5\_block8\_0\_relu[0][0]']** |  | | **conv5\_block8\_1\_bn** | **(BatchNormal (None, 7, 7, 128)** | **512** | **['conv5\_block8\_1\_conv[0][0]']** |  | | **conv5\_block8\_1\_relu** | **(Activatio (None, 7, 7, 128)** | **0** | **['conv5\_block8\_1\_bn[0][0]']** |  | | **conv5\_block8\_2\_conv** | **(Conv2D) (None, 7, 7, 32)** | **36864** | **['conv5\_block8\_1\_relu[0][0]']** |  | | **conv5\_block8\_concat** | **(Concatena (None, 7, 7, 768)** | **0** | **['conv5\_block7\_concat[0][0]',** |  | | **conv5\_block9\_0\_bn** | **(BatchNormal (None, 7, 7, 768)** | **3072** | **['conv5\_block8\_concat[0][0]']** |  | | **conv5\_block9\_0\_relu** | **(Activatio (None, 7, 7, 768)** | **0** | **['conv5\_block9\_0\_bn[0][0]']** |  | | **conv5\_block9\_1\_conv** | **(Conv2D) (None, 7, 7, 128)** | **98304** | **['conv5\_block9\_0\_relu[0][0]']** |  | | **conv5\_block9\_1\_bn** | **(BatchNormal (None, 7, 7, 128)** | **512** | **['conv5\_block9\_1\_conv[0][0]']** |  | | **conv5\_block9\_1\_relu** | **(Activatio (None, 7, 7, 128)** | **0** | **['conv5\_block9\_1\_bn[0][0]']** |  | | **conv5\_block9\_2\_conv** | **(Conv2D) (None, 7, 7, 32)** | **36864** | **['conv5\_block9\_1\_relu[0][0]']** |  | | **conv5\_block9\_concat** | **(Concatena (None, 7, 7, 800)** | **0** | **['conv5\_block8\_concat[0][0]',** |  | | **conv5\_block10\_0\_bn** | **(BatchNorma (None, 7, 7, 800)** | **3200** | **['conv5\_block9\_concat[0][0]']** |  | | **conv5\_block10\_0\_relu** | **(Activati (None, 7, 7, 800)** | **0** | **['conv5\_block10\_0\_bn[0][0]']** |  | | **conv5\_block10\_1\_conv** | **(Conv2D) (None, 7, 7, 128)** | **102400** | **['conv5\_block10\_0\_relu[0][0]']** |  | | **conv5\_block10\_1\_bn** | **(BatchNorma (None, 7, 7, 128)** | **512** | **['conv5\_block10\_1\_conv[0][0]']** |  | | **conv5\_block10\_1\_relu** | **(Activati (None, 7, 7, 128)** | **0** | **['conv5\_block10\_1\_bn[0][0]']** |  | | **conv5\_block10\_2\_conv** | **(Conv2D) (None, 7, 7, 32)** | **36864** | **['conv5\_block10\_1\_relu[0][0]']** |  | | **conv5\_block10\_concat** | **(Concaten (None, 7, 7, 832)** | **0** | **['conv5\_block9\_concat[0][0]',** |  | | **conv5\_block11\_0\_bn** | **(BatchNorma (None, 7, 7, 832)** | **3328** | **['conv5\_block10\_concat[0][0]']** |  | | **conv5\_block11\_0\_relu** | **(Activati (None, 7, 7, 832)** | **0** | **['conv5\_block11\_0\_bn[0][0]']** |  | | **conv5\_block11\_1\_conv** | **(Conv2D) (None, 7, 7, 128)** | **106496** | **['conv5\_block11\_0\_relu[0][0]']** |  | | **conv5\_block11\_1\_bn** | **(BatchNorma (None, 7, 7, 128)** | **512** | **['conv5\_block11\_1\_conv[0][0]']** |  | | **conv5\_block11\_1\_relu** | **(Activati (None, 7, 7, 128)** | **0** | **['conv5\_block11\_1\_bn[0][0]']** |  | | **conv5\_block11\_2\_conv** | **(Conv2D) (None, 7, 7, 32)** | **36864** | **['conv5\_block11\_1\_relu[0][0]']** |  | | **conv5\_block11\_concat** | **(Concaten (None, 7, 7, 864)** | **0** | **['conv5\_block10\_concat[0][0]',** |  | | **conv5\_block12\_0\_bn** | **(BatchNorma (None, 7, 7, 864)** | **3456** | **['conv5\_block11\_concat[0][0]']** |  | | **conv5\_block12\_0\_relu** | **(Activati (None, 7, 7, 864)** | **0** | **['conv5\_block12\_0\_bn[0][0]']** |  | | **conv5\_block12\_1\_conv** | **(Conv2D) (None, 7, 7, 128)** | **110592** | **['conv5\_block12\_0\_relu[0][0]']** |  | | **conv5\_block12\_1\_bn** | **(BatchNorma (None, 7, 7, 128)** | **512** | **['conv5\_block12\_1\_conv[0][0]']** |  | | **conv5\_block12\_1\_relu** | **(Activati (None, 7, 7, 128)** | **0** | **['conv5\_block12\_1\_bn[0][0]']** |  | | **conv5\_block12\_2\_conv** | **(Conv2D) (None, 7, 7, 32)** | **36864** | **['conv5\_block12\_1\_relu[0][0]']** |  | | **conv5\_block12\_concat** | **(Concaten (None, 7, 7, 896)** | **0** | **['conv5\_block11\_concat[0][0]',** |  | | **conv5\_block13\_0\_bn** | **(BatchNorma (None, 7, 7, 896)** | **3584** | **['conv5\_block12\_concat[0][0]']** |  | | **conv5\_block13\_0\_relu** | **(Activati (None, 7, 7, 896)** | **0** | **['conv5\_block13\_0\_bn[0][0]']** |  | | **conv5\_block13\_1\_conv** | **(Conv2D) (None, 7, 7, 128)** | **114688** | **['conv5\_block13\_0\_relu[0][0]']** |  | | **conv5\_block13\_1\_bn** | **(BatchNorma (None, 7, 7, 128)** | **512** | **['conv5\_block13\_1\_conv[0][0]']** |  | | **conv5\_block13\_1\_relu** | **(Activati (None, 7, 7, 128)** | **0** | **['conv5\_block13\_1\_bn[0][0]']** |  | | **conv5\_block13\_2\_conv** | **(Conv2D) (None, 7, 7, 32)** | **36864** | **['conv5\_block13\_1\_relu[0][0]']** |  | | **conv5\_block13\_concat** | **(Concaten (None, 7, 7, 928)** | **0** | **['conv5\_block12\_concat[0][0]',** |  | | **conv5\_block14\_0\_bn** | **(BatchNorma (None, 7, 7, 928)** | **3712** | **['conv5\_block13\_concat[0][0]']** |  | | **conv5\_block14\_0\_relu** | **(Activati (None, 7, 7, 928)** | **0** | **['conv5\_block14\_0\_bn[0][0]']** |  | | **conv5\_block14\_1\_conv** | **(Conv2D) (None, 7, 7, 128)** | **118784** | **['conv5\_block14\_0\_relu[0][0]']** |  | | **conv5\_block14\_1\_bn** | **(BatchNorma (None, 7, 7, 128)** | **512** | **['conv5\_block14\_1\_conv[0][0]']** |  | | **conv5\_block14\_1\_relu** | **(Activati (None, 7, 7, 128)** | **0** | **['conv5\_block14\_1\_bn[0][0]']** |  | | **conv5\_block14\_2\_conv** | **(Conv2D) (None, 7, 7, 32)** | **36864** | **['conv5\_block14\_1\_relu[0][0]']** |  | | **conv5\_block14\_concat** | **(Concaten (None, 7, 7, 960)** | **0** | **['conv5\_block13\_concat[0][0]',** |  | | **conv5\_block15\_0\_bn** | **(BatchNorma (None, 7, 7, 960)** | **3840** | **['conv5\_block14\_concat[0][0]']** |  | | **conv5\_block15\_0\_relu** | **(Activati (None, 7, 7, 960)** | **0** | **['conv5\_block15\_0\_bn[0][0]']** |  | | **conv5\_block15\_1\_conv** | **(Conv2D) (None, 7, 7, 128)** | **122880** | **['conv5\_block15\_0\_relu[0][0]']** |  | | **conv5\_block15\_1\_bn** | **(BatchNorma (None, 7, 7, 128)** | **512** | **['conv5\_block15\_1\_conv[0][0]']** |  | | **conv5\_block15\_1\_relu** | **(Activati (None, 7, 7, 128)** | **0** | **['conv5\_block15\_1\_bn[0][0]']** |  | | **conv5\_block15\_2\_conv** | **(Conv2D) (None, 7, 7, 32)** | **36864** | **['conv5\_block15\_1\_relu[0][0]']** |  | | **conv5\_block15\_concat** | **(Concaten (None, 7, 7, 992)** | **0** | **['conv5\_block14\_concat[0][0]',** |  | | **conv5\_block16\_0\_bn** | **(BatchNorma (None, 7, 7, 992)** | **3968** | **['conv5\_block15\_concat[0][0]']** |  | | **conv5\_block16\_0\_relu** | **(Activati (None, 7, 7, 992)** | **0** | **['conv5\_block16\_0\_bn[0][0]']** |  | | **conv5\_block16\_1\_conv** | **(Conv2D) (None, 7, 7, 128)** | **126976** | **['conv5\_block16\_0\_relu[0][0]']** |  | | **conv5\_block16\_1\_bn** | **(BatchNorma (None, 7, 7, 128)** | **512** | **['conv5\_block16\_1\_conv[0][0]']** |  | | **conv5\_block16\_1\_relu** | **(Activati (None, 7, 7, 128)** | **0** | **['conv5\_block16\_1\_bn[0][0]']** |  | | **conv5\_block16\_2\_conv** | **(Conv2D) (None, 7, 7, 32)** | **36864** | **['conv5\_block16\_1\_relu[0][0]']** |  | | **conv5\_block16\_concat** | **(Concaten (None, 7, 7, 1024)** | **0** | **['conv5\_block15\_concat[0][0]',** |  | | **bn** | **(BatchNormalization) (None, 7, 7, 1024)** | **4096** | **['conv5\_block16\_concat[0][0]']** |  | | **relu** | **(Activation) (None, 7, 7, 1024)** | **0** | **['bn[0][0]']** |  | | **avg\_pool** | **(GlobalAveragePooling (None, 1024)** | **0** | **['relu[0][0]']** |  | | **predictions** | **(Dense) (None, 1000)** | **1025000** | **['avg\_pool[0][0]']** |  | | |  |  |  |  | | --- | --- | --- | --- | | **Layer (type)** | **Output Shape** | **Parameters** | **Connected to** | |  | | input\_2 | (InputLayer) [(None, 224, 224, 3 | 0 | [] |  | | rescaling | (Rescaling) (None, 224, 224, 3) | 0 | ['input\_2[0][0]'] |  | | normalization | (Normalization) (None, 224, 224, 3) | 7 | ['rescaling[0][0]'] |  | | stem\_conv\_pad | (ZeroPadding2D) (None, 225, 225, 3) | 0 | ['normalization[0][0]'] |  | | stem\_conv | (Conv2D) (None, 112, 112, 32 | 864 | ['stem\_conv\_pad[0][0]'] |  | | stem\_bn | (BatchNormalization) (None, 112, 112, 32 | 128 | ['stem\_conv[0][0]'] |  | | stem\_activation | (Activation) (None, 112, 112, 32 | 0 | ['stem\_bn[0][0]'] |  | | block1a\_dwconv | (DepthwiseConv2 (None, 112, 112, 32 | 288 | ['stem\_activation[0][0]'] |  | | block1a\_bn | (BatchNormalization (None, 112, 112, 32 | 128 | ['block1a\_dwconv[0][0]'] |  | | block1a\_activation | (Activation (None, 112, 112, 32 | 0 | ['block1a\_bn[0][0]'] |  | | block1a\_se\_squeeze | (GlobalAver (None, 32) | 0 | ['block1a\_activation[0][0]'] |  | | block1a\_se\_reshape | (Reshape) (None, 1, 1, 32) | 0 | ['block1a\_se\_squeeze[0][0]'] |  | | block1a\_se\_reduce | (Conv2D) (None, 1, 1, 8) | 264 | ['block1a\_se\_reshape[0][0]'] |  | | block1a\_se\_expand | (Conv2D) (None, 1, 1, 32) | 288 | ['block1a\_se\_reduce[0][0]'] |  | | block1a\_se\_excite | (Multiply) (None, 112, 112, 32 | 0 | ['block1a\_activation[0][0]', |  | | block1a\_project\_conv | (Conv2D) (None, 112, 112, 16 | 512 | ['block1a\_se\_excite[0][0]'] |  | | block1a\_project\_bn | (BatchNorma (None, 112, 112, 16 | 64 | ['block1a\_project\_conv[0][0]'] |  | | block2a\_expand\_conv | (Conv2D) (None, 112, 112, 96 | 1536 | ['block1a\_project\_bn[0][0]'] |  | | block2a\_expand\_bn | (BatchNormal (None, 112, 112, 96 | 384 | ['block2a\_expand\_conv[0][0]'] |  | | block2a\_expand\_activation | (Act (None, 112, 112, 96 | 0 | ['block2a\_expand\_bn[0][0]'] |  | | block2a\_dwconv\_pad | (ZeroPaddin (None, 113, 113, 96 | 0 | ['block2a\_expand\_activation[0][0] |  | | block2a\_dwconv | (DepthwiseConv2 (None, 56, 56, 96) | 864 | ['block2a\_dwconv\_pad[0][0]'] |  | | block2a\_bn | (BatchNormalization (None, 56, 56, 96) | 384 | ['block2a\_dwconv[0][0]'] |  | | block2a\_activation | (Activation (None, 56, 56, 96) | 0 | ['block2a\_bn[0][0]'] |  | | block2a\_se\_squeeze | (GlobalAver (None, 96) | 0 | ['block2a\_activation[0][0]'] |  | | block2a\_se\_reshape | (Reshape) (None, 1, 1, 96) | 0 | ['block2a\_se\_squeeze[0][0]'] |  | | block2a\_se\_reduce | (Conv2D) (None, 1, 1, 4) | 388 | ['block2a\_se\_reshape[0][0]'] |  | | block2a\_se\_expand | (Conv2D) (None, 1, 1, 96) | 480 | ['block2a\_se\_reduce[0][0]'] |  | | block2a\_se\_excite | (Multiply) (None, 56, 56, 96) | 0 | ['block2a\_activation[0][0]', |  | | block2a\_project\_conv | (Conv2D) (None, 56, 56, 24) | 2304 | ['block2a\_se\_excite[0][0]'] |  | | block2a\_project\_bn | (BatchNorma (None, 56, 56, 24) | 96 | ['block2a\_project\_conv[0][0]'] |  | | block2b\_expand\_conv | (Conv2D) (None, 56, 56, 144) | 3456 | ['block2a\_project\_bn[0][0]'] |  | | block2b\_expand\_bn | (BatchNormal (None, 56, 56, 144) | 576 | ['block2b\_expand\_conv[0][0]'] |  | | block2b\_expand\_activation | (Act (None, 56, 56, 144) | 0 | ['block2b\_expand\_bn[0][0]'] |  | | block2b\_dwconv | (DepthwiseConv2 (None, 56, 56, 144) | 1296 | ['block2b\_expand\_activation[0][0] |  | | block2b\_bn | (BatchNormalization (None, 56, 56, 144) | 576 | ['block2b\_dwconv[0][0]'] |  | | block2b\_activation | (Activation (None, 56, 56, 144) | 0 | ['block2b\_bn[0][0]'] |  | | block2b\_se\_squeeze | (GlobalAver (None, 144) | 0 | ['block2b\_activation[0][0]'] |  | | block2b\_se\_reshape | (Reshape) (None, 1, 1, 144) | 0 | ['block2b\_se\_squeeze[0][0]'] |  | | block2b\_se\_reduce | (Conv2D) (None, 1, 1, 6) | 870 | ['block2b\_se\_reshape[0][0]'] |  | | block2b\_se\_expand | (Conv2D) (None, 1, 1, 144) | 1008 | ['block2b\_se\_reduce[0][0]'] |  | | block2b\_se\_excite | (Multiply) (None, 56, 56, 144) | 0 | ['block2b\_activation[0][0]', |  | | block2b\_project\_conv | (Conv2D) (None, 56, 56, 24) | 3456 | ['block2b\_se\_excite[0][0]'] |  | | block2b\_project\_bn | (BatchNorma (None, 56, 56, 24) | 96 | ['block2b\_project\_conv[0][0]'] |  | | block2b\_drop | (Dropout) (None, 56, 56, 24) | 0 | ['block2b\_project\_bn[0][0]'] |  | | block2b\_add | (Add) (None, 56, 56, 24) | 0 | ['block2b\_drop[0][0]', |  | | block3a\_expand\_conv | (Conv2D) (None, 56, 56, 144) | 3456 | ['block2b\_add[0][0]'] |  | | block3a\_expand\_bn | (BatchNormal (None, 56, 56, 144) | 576 | ['block3a\_expand\_conv[0][0]'] |  | | block3a\_expand\_activation | (Act (None, 56, 56, 144) | 0 | ['block3a\_expand\_bn[0][0]'] |  | | block3a\_dwconv\_pad | (ZeroPaddin (None, 59, 59, 144) | 0 | ['block3a\_expand\_activation[0][0] |  | | block3a\_dwconv | (DepthwiseConv2 (None, 28, 28, 144) | 3600 | ['block3a\_dwconv\_pad[0][0]'] |  | | block3a\_bn | (BatchNormalization (None, 28, 28, 144) | 576 | ['block3a\_dwconv[0][0]'] |  | | block3a\_activation | (Activation (None, 28, 28, 144) | 0 | ['block3a\_bn[0][0]'] |  | | block3a\_se\_squeeze | (GlobalAver (None, 144) | 0 | ['block3a\_activation[0][0]'] |  | | block3a\_se\_reshape | (Reshape) (None, 1, 1, 144) | 0 | ['block3a\_se\_squeeze[0][0]'] |  | | block3a\_se\_reduce | (Conv2D) (None, 1, 1, 6) | 870 | ['block3a\_se\_reshape[0][0]'] |  | | block3a\_se\_expand | (Conv2D) (None, 1, 1, 144) | 1008 | ['block3a\_se\_reduce[0][0]'] |  | | block3a\_se\_excite | (Multiply) (None, 28, 28, 144) | 0 | ['block3a\_activation[0][0]', |  | | block3a\_project\_conv | (Conv2D) (None, 28, 28, 40) | 5760 | ['block3a\_se\_excite[0][0]'] |  | | block3a\_project\_bn | (BatchNorma (None, 28, 28, 40) | 160 | ['block3a\_project\_conv[0][0]'] |  | | block3b\_expand\_conv | (Conv2D) (None, 28, 28, 240) | 9600 | ['block3a\_project\_bn[0][0]'] |  | | block3b\_expand\_bn | (BatchNormal (None, 28, 28, 240) | 960 | ['block3b\_expand\_conv[0][0]'] |  | | block3b\_expand\_activation | (Act (None, 28, 28, 240) | 0 | ['block3b\_expand\_bn[0][0]'] |  | | block3b\_dwconv | (DepthwiseConv2 (None, 28, 28, 240) | 6000 | ['block3b\_expand\_activation[0][0] |  | | block3b\_bn | (BatchNormalization (None, 28, 28, 240) | 960 | ['block3b\_dwconv[0][0]'] |  | | block3b\_activation | (Activation (None, 28, 28, 240) | 0 | ['block3b\_bn[0][0]'] |  | | block3b\_se\_squeeze | (GlobalAver (None, 240) | 0 | ['block3b\_activation[0][0]'] |  | | block3b\_se\_reshape | (Reshape) (None, 1, 1, 240) | 0 | ['block3b\_se\_squeeze[0][0]'] |  | | block3b\_se\_reduce | (Conv2D) (None, 1, 1, 10) | 2410 | ['block3b\_se\_reshape[0][0]'] |  | | block3b\_se\_expand | (Conv2D) (None, 1, 1, 240) | 2640 | ['block3b\_se\_reduce[0][0]'] |  | | block3b\_se\_excite | (Multiply) (None, 28, 28, 240) | 0 | ['block3b\_activation[0][0]', |  | | block3b\_project\_conv | (Conv2D) (None, 28, 28, 40) | 9600 | ['block3b\_se\_excite[0][0]'] |  | | block3b\_project\_bn | (BatchNorma (None, 28, 28, 40) | 160 | ['block3b\_project\_conv[0][0]'] |  | | block3b\_drop | (Dropout) (None, 28, 28, 40) | 0 | ['block3b\_project\_bn[0][0]'] |  | | block3b\_add | (Add) (None, 28, 28, 40) | 0 | ['block3b\_drop[0][0]', |  | | block4a\_expand\_conv | (Conv2D) (None, 28, 28, 240) | 9600 | ['block3b\_add[0][0]'] |  | | block4a\_expand\_bn | (BatchNormal (None, 28, 28, 240) | 960 | ['block4a\_expand\_conv[0][0]'] |  | | block4a\_expand\_activation | (Act (None, 28, 28, 240) | 0 | ['block4a\_expand\_bn[0][0]'] |  | | block4a\_dwconv\_pad | (ZeroPaddin (None, 29, 29, 240) | 0 | ['block4a\_expand\_activation[0][0] |  | | block4a\_dwconv | (DepthwiseConv2 (None, 14, 14, 240) | 2160 | ['block4a\_dwconv\_pad[0][0]'] |  | | block4a\_bn | (BatchNormalization (None, 14, 14, 240) | 960 | ['block4a\_dwconv[0][0]'] |  | | block4a\_activation | (Activation (None, 14, 14, 240) | 0 | ['block4a\_bn[0][0]'] |  | | block4a\_se\_squeeze | (GlobalAver (None, 240) | 0 | ['block4a\_activation[0][0]'] |  | | block4a\_se\_reshape | (Reshape) (None, 1, 1, 240) | 0 | ['block4a\_se\_squeeze[0][0]'] |  | | block4a\_se\_reduce | (Conv2D) (None, 1, 1, 10) | 2410 | ['block4a\_se\_reshape[0][0]'] |  | | block4a\_se\_expand | (Conv2D) (None, 1, 1, 240) | 2640 | ['block4a\_se\_reduce[0][0]'] |  | | block4a\_se\_excite | (Multiply) (None, 14, 14, 240) | 0 | ['block4a\_activation[0][0]', |  | | block4a\_project\_conv | (Conv2D) (None, 14, 14, 80) | 19200 | ['block4a\_se\_excite[0][0]'] |  | | block4a\_project\_bn | (BatchNorma (None, 14, 14, 80) | 320 | ['block4a\_project\_conv[0][0]'] |  | | block4b\_expand\_conv | (Conv2D) (None, 14, 14, 480) | 38400 | ['block4a\_project\_bn[0][0]'] |  | | block4b\_expand\_bn | (BatchNormal (None, 14, 14, 480) | 1920 | ['block4b\_expand\_conv[0][0]'] |  | | block4b\_expand\_activation | (Act (None, 14, 14, 480) | 0 | ['block4b\_expand\_bn[0][0]'] |  | | block4b\_dwconv | (DepthwiseConv2 (None, 14, 14, 480) | 4320 | ['block4b\_expand\_activation[0][0] |  | | block4b\_bn | (BatchNormalization (None, 14, 14, 480) | 1920 | ['block4b\_dwconv[0][0]'] |  | | block4b\_activation | (Activation (None, 14, 14, 480) | 0 | ['block4b\_bn[0][0]'] |  | | block4b\_se\_squeeze | (GlobalAver (None, 480) | 0 | ['block4b\_activation[0][0]'] |  | | block4b\_se\_reshape | (Reshape) (None, 1, 1, 480) | 0 | ['block4b\_se\_squeeze[0][0]'] |  | | block4b\_se\_reduce | (Conv2D) (None, 1, 1, 20) | 9620 | ['block4b\_se\_reshape[0][0]'] |  | | block4b\_se\_expand | (Conv2D) (None, 1, 1, 480) | 10080 | ['block4b\_se\_reduce[0][0]'] |  | | block4b\_se\_excite | (Multiply) (None, 14, 14, 480) | 0 | ['block4b\_activation[0][0]', |  | | block4b\_project\_conv | (Conv2D) (None, 14, 14, 80) | 38400 | ['block4b\_se\_excite[0][0]'] |  | | block4b\_project\_bn | (BatchNorma (None, 14, 14, 80) | 320 | ['block4b\_project\_conv[0][0]'] |  | | block4b\_drop | (Dropout) (None, 14, 14, 80) | 0 | ['block4b\_project\_bn[0][0]'] |  | | block4b\_add | (Add) (None, 14, 14, 80) | 0 | ['block4b\_drop[0][0]', |  | | block4c\_expand\_conv | (Conv2D) (None, 14, 14, 480) | 38400 | ['block4b\_add[0][0]'] |  | | block4c\_expand\_bn | (BatchNormal (None, 14, 14, 480) | 1920 | ['block4c\_expand\_conv[0][0]'] |  | | block4c\_expand\_activation | (Act (None, 14, 14, 480) | 0 | ['block4c\_expand\_bn[0][0]'] |  | | block4c\_dwconv | (DepthwiseConv2 (None, 14, 14, 480) | 4320 | ['block4c\_expand\_activation[0][0] |  | | block4c\_bn | (BatchNormalization (None, 14, 14, 480) | 1920 | ['block4c\_dwconv[0][0]'] |  | | block4c\_activation | (Activation (None, 14, 14, 480) | 0 | ['block4c\_bn[0][0]'] |  | | block4c\_se\_squeeze | (GlobalAver (None, 480) | 0 | ['block4c\_activation[0][0]'] |  | | block4c\_se\_reshape | (Reshape) (None, 1, 1, 480) | 0 | ['block4c\_se\_squeeze[0][0]'] |  | | block4c\_se\_reduce | (Conv2D) (None, 1, 1, 20) | 9620 | ['block4c\_se\_reshape[0][0]'] |  | | block4c\_se\_expand | (Conv2D) (None, 1, 1, 480) | 10080 | ['block4c\_se\_reduce[0][0]'] |  | | block4c\_se\_excite | (Multiply) (None, 14, 14, 480) | 0 | ['block4c\_activation[0][0]', |  | | block4c\_project\_conv | (Conv2D) (None, 14, 14, 80) | 38400 | ['block4c\_se\_excite[0][0]'] |  | | block4c\_project\_bn | (BatchNorma (None, 14, 14, 80) | 320 | ['block4c\_project\_conv[0][0]'] |  | | block4c\_drop | (Dropout) (None, 14, 14, 80) | 0 | ['block4c\_project\_bn[0][0]'] |  | | block4c\_add | (Add) (None, 14, 14, 80) | 0 | ['block4c\_drop[0][0]', |  | | block5a\_expand\_conv | (Conv2D) (None, 14, 14, 480) | 38400 | ['block4c\_add[0][0]'] |  | | block5a\_expand\_bn | (BatchNormal (None, 14, 14, 480) | 1920 | ['block5a\_expand\_conv[0][0]'] |  | | block5a\_expand\_activation | (Act (None, 14, 14, 480) | 0 | ['block5a\_expand\_bn[0][0]'] |  | | block5a\_dwconv | (DepthwiseConv2 (None, 14, 14, 480) | 12000 | ['block5a\_expand\_activation[0][0] |  | | block5a\_bn | (BatchNormalization (None, 14, 14, 480) | 1920 | ['block5a\_dwconv[0][0]'] |  | | block5a\_activation | (Activation (None, 14, 14, 480) | 0 | ['block5a\_bn[0][0]'] |  | | block5a\_se\_squeeze | (GlobalAver (None, 480) | 0 | ['block5a\_activation[0][0]'] |  | | block5a\_se\_reshape | (Reshape) (None, 1, 1, 480) | 0 | ['block5a\_se\_squeeze[0][0]'] |  | | block5a\_se\_reduce | (Conv2D) (None, 1, 1, 20) | 9620 | ['block5a\_se\_reshape[0][0]'] |  | | block5a\_se\_expand | (Conv2D) (None, 1, 1, 480) | 10080 | ['block5a\_se\_reduce[0][0]'] |  | | block5a\_se\_excite | (Multiply) (None, 14, 14, 480) | 0 | ['block5a\_activation[0][0]', |  | | block5a\_project\_conv | (Conv2D) (None, 14, 14, 112) | 53760 | ['block5a\_se\_excite[0][0]'] |  | | block5a\_project\_bn | (BatchNorma (None, 14, 14, 112) | 448 | ['block5a\_project\_conv[0][0]'] |  | | block5b\_expand\_conv | (Conv2D) (None, 14, 14, 672) | 75264 | ['block5a\_project\_bn[0][0]'] |  | | block5b\_expand\_bn | (BatchNormal (None, 14, 14, 672) | 2688 | ['block5b\_expand\_conv[0][0]'] |  | | block5b\_expand\_activation | (Act (None, 14, 14, 672) | 0 | ['block5b\_expand\_bn[0][0]'] |  | | block5b\_dwconv | (DepthwiseConv2 (None, 14, 14, 672) | 16800 | ['block5b\_expand\_activation[0][0] |  | | block5b\_bn | (BatchNormalization (None, 14, 14, 672) | 2688 | ['block5b\_dwconv[0][0]'] |  | | block5b\_activation | (Activation (None, 14, 14, 672) | 0 | ['block5b\_bn[0][0]'] |  | | block5b\_se\_squeeze | (GlobalAver (None, 672) | 0 | ['block5b\_activation[0][0]'] |  | | block5b\_se\_reshape | (Reshape) (None, 1, 1, 672) | 0 | ['block5b\_se\_squeeze[0][0]'] |  | | block5b\_se\_reduce | (Conv2D) (None, 1, 1, 28) | 18844 | ['block5b\_se\_reshape[0][0]'] |  | | block5b\_se\_expand | (Conv2D) (None, 1, 1, 672) | 19488 | ['block5b\_se\_reduce[0][0]'] |  | | block5b\_se\_excite | (Multiply) (None, 14, 14, 672) | 0 | ['block5b\_activation[0][0]', |  | | block5b\_project\_conv | (Conv2D) (None, 14, 14, 112) | 75264 | ['block5b\_se\_excite[0][0]'] |  | | block5b\_project\_bn | (BatchNorma (None, 14, 14, 112) | 448 | ['block5b\_project\_conv[0][0]'] |  | | block5b\_drop | (Dropout) (None, 14, 14, 112) | 0 | ['block5b\_project\_bn[0][0]'] |  | | block5b\_add | (Add) (None, 14, 14, 112) | 0 | ['block5b\_drop[0][0]', |  | | block5c\_expand\_conv | (Conv2D) (None, 14, 14, 672) | 75264 | ['block5b\_add[0][0]'] |  | | block5c\_expand\_bn | (BatchNormal (None, 14, 14, 672) | 2688 | ['block5c\_expand\_conv[0][0]'] |  | | block5c\_expand\_activation | (Act (None, 14, 14, 672) | 0 | ['block5c\_expand\_bn[0][0]'] |  | | block5c\_dwconv | (DepthwiseConv2 (None, 14, 14, 672) | 16800 | ['block5c\_expand\_activation[0][0] |  | | block5c\_bn | (BatchNormalization (None, 14, 14, 672) | 2688 | ['block5c\_dwconv[0][0]'] |  | | block5c\_activation | (Activation (None, 14, 14, 672) | 0 | ['block5c\_bn[0][0]'] |  | | block5c\_se\_squeeze | (GlobalAver (None, 672) | 0 | ['block5c\_activation[0][0]'] |  | | block5c\_se\_reshape | (Reshape) (None, 1, 1, 672) | 0 | ['block5c\_se\_squeeze[0][0]'] |  | | block5c\_se\_reduce | (Conv2D) (None, 1, 1, 28) | 18844 | ['block5c\_se\_reshape[0][0]'] |  | | block5c\_se\_expand | (Conv2D) (None, 1, 1, 672) | 19488 | ['block5c\_se\_reduce[0][0]'] |  | | block5c\_se\_excite | (Multiply) (None, 14, 14, 672) | 0 | ['block5c\_activation[0][0]', |  | | block5c\_project\_conv | (Conv2D) (None, 14, 14, 112) | 75264 | ['block5c\_se\_excite[0][0]'] |  | | block5c\_project\_bn | (BatchNorma (None, 14, 14, 112) | 448 | ['block5c\_project\_conv[0][0]'] |  | | block5c\_drop | (Dropout) (None, 14, 14, 112) | 0 | ['block5c\_project\_bn[0][0]'] |  | | block5c\_add | (Add) (None, 14, 14, 112) | 0 | ['block5c\_drop[0][0]', |  | | block6a\_expand\_conv | (Conv2D) (None, 14, 14, 672) | 75264 | ['block5c\_add[0][0]'] |  | | block6a\_expand\_bn | (BatchNormal (None, 14, 14, 672) | 2688 | ['block6a\_expand\_conv[0][0]'] |  | | block6a\_expand\_activation | (Act (None, 14, 14, 672) | 0 | ['block6a\_expand\_bn[0][0]'] |  | | block6a\_dwconv\_pad | (ZeroPaddin (None, 17, 17, 672) | 0 | ['block6a\_expand\_activation[0][0] |  | | block6a\_dwconv | (DepthwiseConv2 (None, 7, 7, 672) | 16800 | ['block6a\_dwconv\_pad[0][0]'] |  | | block6a\_bn | (BatchNormalization (None, 7, 7, 672) | 2688 | ['block6a\_dwconv[0][0]'] |  | | block6a\_activation | (Activation (None, 7, 7, 672) | 0 | ['block6a\_bn[0][0]'] |  | | block6a\_se\_squeeze | (GlobalAver (None, 672) | 0 | ['block6a\_activation[0][0]'] |  | | block6a\_se\_reshape | (Reshape) (None, 1, 1, 672) | 0 | ['block6a\_se\_squeeze[0][0]'] |  | | block6a\_se\_reduce | (Conv2D) (None, 1, 1, 28) | 18844 | ['block6a\_se\_reshape[0][0]'] |  | | block6a\_se\_expand | (Conv2D) (None, 1, 1, 672) | 19488 | ['block6a\_se\_reduce[0][0]'] |  | | block6a\_se\_excite | (Multiply) (None, 7, 7, 672) | 0 | ['block6a\_activation[0][0]', |  | | block6a\_project\_conv | (Conv2D) (None, 7, 7, 192) | 129024 | ['block6a\_se\_excite[0][0]'] |  | | block6a\_project\_bn | (BatchNorma (None, 7, 7, 192) | 768 | ['block6a\_project\_conv[0][0]'] |  | | block6b\_expand\_conv | (Conv2D) (None, 7, 7, 1152) | 221184 | ['block6a\_project\_bn[0][0]'] |  | | block6b\_expand\_bn | (BatchNormal (None, 7, 7, 1152) | 4608 | ['block6b\_expand\_conv[0][0]'] |  | | block6b\_expand\_activation | (Act (None, 7, 7, 1152) | 0 | ['block6b\_expand\_bn[0][0]'] |  | | block6b\_dwconv | (DepthwiseConv2 (None, 7, 7, 1152) | 28800 | ['block6b\_expand\_activation[0][0] |  | | block6b\_bn | (BatchNormalization (None, 7, 7, 1152) | 4608 | ['block6b\_dwconv[0][0]'] |  | | block6b\_activation | (Activation (None, 7, 7, 1152) | 0 | ['block6b\_bn[0][0]'] |  | | block6b\_se\_squeeze | (GlobalAver (None, 1152) | 0 | ['block6b\_activation[0][0]'] |  | | block6b\_se\_reshape | (Reshape) (None, 1, 1, 1152) | 0 | ['block6b\_se\_squeeze[0][0]'] |  | | block6b\_se\_reduce | (Conv2D) (None, 1, 1, 48) | 55344 | ['block6b\_se\_reshape[0][0]'] |  | | block6b\_se\_expand | (Conv2D) (None, 1, 1, 1152) | 56448 | ['block6b\_se\_reduce[0][0]'] |  | | block6b\_se\_excite | (Multiply) (None, 7, 7, 1152) | 0 | ['block6b\_activation[0][0]', |  | | block6b\_project\_conv | (Conv2D) (None, 7, 7, 192) | 221184 | ['block6b\_se\_excite[0][0]'] |  | | block6b\_project\_bn | (BatchNorma (None, 7, 7, 192) | 768 | ['block6b\_project\_conv[0][0]'] |  | | block6b\_drop | (Dropout) (None, 7, 7, 192) | 0 | ['block6b\_project\_bn[0][0]'] |  | | block6b\_add | (Add) (None, 7, 7, 192) | 0 | ['block6b\_drop[0][0]', |  | | block6c\_expand\_conv | (Conv2D) (None, 7, 7, 1152) | 221184 | ['block6b\_add[0][0]'] |  | | block6c\_expand\_bn | (BatchNormal (None, 7, 7, 1152) | 4608 | ['block6c\_expand\_conv[0][0]'] |  | | block6c\_expand\_activation | (Act (None, 7, 7, 1152) | 0 | ['block6c\_expand\_bn[0][0]'] |  | | block6c\_dwconv | (DepthwiseConv2 (None, 7, 7, 1152) | 28800 | ['block6c\_expand\_activation[0][0] |  | | block6c\_bn | (BatchNormalization (None, 7, 7, 1152) | 4608 | ['block6c\_dwconv[0][0]'] |  | | block6c\_activation | (Activation (None, 7, 7, 1152) | 0 | ['block6c\_bn[0][0]'] |  | | block6c\_se\_squeeze | (GlobalAver (None, 1152) | 0 | ['block6c\_activation[0][0]'] |  | | block6c\_se\_reshape | (Reshape) (None, 1, 1, 1152) | 0 | ['block6c\_se\_squeeze[0][0]'] |  | | block6c\_se\_reduce | (Conv2D) (None, 1, 1, 48) | 55344 | ['block6c\_se\_reshape[0][0]'] |  | | block6c\_se\_expand | (Conv2D) (None, 1, 1, 1152) | 56448 | ['block6c\_se\_reduce[0][0]'] |  | | block6c\_se\_excite | (Multiply) (None, 7, 7, 1152) | 0 | ['block6c\_activation[0][0]', |  | | block6c\_project\_conv | (Conv2D) (None, 7, 7, 192) | 221184 | ['block6c\_se\_excite[0][0]'] |  | | block6c\_project\_bn | (BatchNorma (None, 7, 7, 192) | 768 | ['block6c\_project\_conv[0][0]'] |  | | block6c\_drop | (Dropout) (None, 7, 7, 192) | 0 | ['block6c\_project\_bn[0][0]'] |  | | block6c\_add | (Add) (None, 7, 7, 192) | 0 | ['block6c\_drop[0][0]', |  | | block6d\_expand\_conv | (Conv2D) (None, 7, 7, 1152) | 221184 | ['block6c\_add[0][0]'] |  | | block6d\_expand\_bn | (BatchNormal (None, 7, 7, 1152) | 4608 | ['block6d\_expand\_conv[0][0]'] |  | | block6d\_expand\_activation | (Act (None, 7, 7, 1152) | 0 | ['block6d\_expand\_bn[0][0]'] |  | | block6d\_dwconv | (DepthwiseConv2 (None, 7, 7, 1152) | 28800 | ['block6d\_expand\_activation[0][0] |  | | block6d\_bn | (BatchNormalization (None, 7, 7, 1152) | 4608 | ['block6d\_dwconv[0][0]'] |  | | block6d\_activation | (Activation (None, 7, 7, 1152) | 0 | ['block6d\_bn[0][0]'] |  | | block6d\_se\_squeeze | (GlobalAver (None, 1152) | 0 | ['block6d\_activation[0][0]'] |  | | block6d\_se\_reshape | (Reshape) (None, 1, 1, 1152) | 0 | ['block6d\_se\_squeeze[0][0]'] |  | | block6d\_se\_reduce | (Conv2D) (None, 1, 1, 48) | 55344 | ['block6d\_se\_reshape[0][0]'] |  | | block6d\_se\_expand | (Conv2D) (None, 1, 1, 1152) | 56448 | ['block6d\_se\_reduce[0][0]'] |  | | block6d\_se\_excite | (Multiply) (None, 7, 7, 1152) | 0 | ['block6d\_activation[0][0]', |  | | block6d\_project\_conv | (Conv2D) (None, 7, 7, 192) | 221184 | ['block6d\_se\_excite[0][0]'] |  | | block6d\_project\_bn | (BatchNorma (None, 7, 7, 192) | 768 | ['block6d\_project\_conv[0][0]'] |  | | block6d\_drop | (Dropout) (None, 7, 7, 192) | 0 | ['block6d\_project\_bn[0][0]'] |  | | block6d\_add | (Add) (None, 7, 7, 192) | 0 | ['block6d\_drop[0][0]', |  | | block7a\_expand\_conv | (Conv2D) (None, 7, 7, 1152) | 221184 | ['block6d\_add[0][0]'] |  | | block7a\_expand\_bn | (BatchNormal (None, 7, 7, 1152) | 4608 | ['block7a\_expand\_conv[0][0]'] |  | | block7a\_expand\_activation | (Act (None, 7, 7, 1152) | 0 | ['block7a\_expand\_bn[0][0]'] |  | | block7a\_dwconv | (DepthwiseConv2 (None, 7, 7, 1152) | 10368 | ['block7a\_expand\_activation[0][0] |  | | block7a\_bn | (BatchNormalization (None, 7, 7, 1152) | 4608 | ['block7a\_dwconv[0][0]'] |  | | block7a\_activation | (Activation (None, 7, 7, 1152) | 0 | ['block7a\_bn[0][0]'] |  | | block7a\_se\_squeeze | (GlobalAver (None, 1152) | 0 | ['block7a\_activation[0][0]'] |  | | block7a\_se\_reshape | (Reshape) (None, 1, 1, 1152) | 0 | ['block7a\_se\_squeeze[0][0]'] |  | | block7a\_se\_reduce | (Conv2D) (None, 1, 1, 48) | 55344 | ['block7a\_se\_reshape[0][0]'] |  | | block7a\_se\_expand | (Conv2D) (None, 1, 1, 1152) | 56448 | ['block7a\_se\_reduce[0][0]'] |  | | block7a\_se\_excite | (Multiply) (None, 7, 7, 1152) | 0 | ['block7a\_activation[0][0]', |  | | block7a\_project\_conv | (Conv2D) (None, 7, 7, 320) | 368640 | ['block7a\_se\_excite[0][0]'] |  | | block7a\_project\_bn | (BatchNorma (None, 7, 7, 320) | 1280 | ['block7a\_project\_conv[0][0]'] |  | | top\_conv | (Conv2D) (None, 7, 7, 1280) | 409600 | ['block7a\_project\_bn[0][0]'] |  | | top\_bn | (BatchNormalization) (None, 7, 7, 1280) | 5120 | ['top\_conv[0][0]'] |  | | top\_activation | (Activation) (None, 7, 7, 1280) | 0 | ['top\_bn[0][0]'] |  | | avg\_pool | (GlobalAveragePooling (None, 1280) | 0 | ['top\_activation[0][0]'] |  | | top\_dropout | (Dropout) (None, 1280) | 0 | ['avg\_pool[0][0]'] |  | | predictions | (Dense) (None, 1000) | 1281000 | ['top\_dropout[0][0]'] |  | |
| **ResNet50 Summary** |
| |  |  |  |  | | --- | --- | --- | --- | | **Layer (type)** | **Output Shape** | **Parameters** | **Connected to** | |  | | input\_3 | (InputLayer) [(None, 224, 224, 3 | 0 | [] |  | | conv1\_pad | (ZeroPadding2D) (None, 230, 230, 3) | 0 | ['input\_3[0][0]'] |  | | conv1\_conv | (Conv2D) (None, 112, 112, 64 | 9472 | ['conv1\_pad[0][0]'] |  | | conv1\_bn | (BatchNormalization) (None, 112, 112, 64 | 256 | ['conv1\_conv[0][0]'] |  | | conv1\_relu | (Activation) (None, 112, 112, 64 | 0 | ['conv1\_bn[0][0]'] |  | | pool1\_pad | (ZeroPadding2D) (None, 114, 114, 64 | 0 | ['conv1\_relu[0][0]'] |  | | pool1\_pool | (MaxPooling2D) (None, 56, 56, 64) | 0 | ['pool1\_pad[0][0]'] |  | | conv2\_block1\_1\_conv | (Conv2D) (None, 56, 56, 64) | 4160 | ['pool1\_pool[0][0]'] |  | | conv2\_block1\_1\_bn | (BatchNormal (None, 56, 56, 64) | 256 | ['conv2\_block1\_1\_conv[0][0]'] |  | | conv2\_block1\_1\_relu | (Activatio (None, 56, 56, 64) | 0 | ['conv2\_block1\_1\_bn[0][0]'] |  | | conv2\_block1\_2\_conv | (Conv2D) (None, 56, 56, 64) | 36928 | ['conv2\_block1\_1\_relu[0][0]'] |  | | conv2\_block1\_2\_bn | (BatchNormal (None, 56, 56, 64) | 256 | ['conv2\_block1\_2\_conv[0][0]'] |  | | conv2\_block1\_2\_relu | (Activatio (None, 56, 56, 64) | 0 | ['conv2\_block1\_2\_bn[0][0]'] |  | | conv2\_block1\_0\_conv | (Conv2D) (None, 56, 56, 256) | 16640 | ['pool1\_pool[0][0]'] |  | | conv2\_block1\_3\_conv | (Conv2D) (None, 56, 56, 256) | 16640 | ['conv2\_block1\_2\_relu[0][0]'] |  | | conv2\_block1\_0\_bn | (BatchNormal (None, 56, 56, 256) | 1024 | ['conv2\_block1\_0\_conv[0][0]'] |  | | conv2\_block1\_3\_bn | (BatchNormal (None, 56, 56, 256) | 1024 | ['conv2\_block1\_3\_conv[0][0]'] |  | | conv2\_block1\_add | (Add) (None, 56, 56, 256) | 0 | ['conv2\_block1\_0\_bn[0][0]', |  | | conv2\_block1\_out | (Activation) (None, 56, 56, 256) | 0 | ['conv2\_block1\_add[0][0]'] |  | | conv2\_block2\_1\_conv | (Conv2D) (None, 56, 56, 64) | 16448 | ['conv2\_block1\_out[0][0]'] |  | | conv2\_block2\_1\_bn | (BatchNormal (None, 56, 56, 64) | 256 | ['conv2\_block2\_1\_conv[0][0]'] |  | | conv2\_block2\_1\_relu | (Activatio (None, 56, 56, 64) | 0 | ['conv2\_block2\_1\_bn[0][0]'] |  | | conv2\_block2\_2\_conv | (Conv2D) (None, 56, 56, 64) | 36928 | ['conv2\_block2\_1\_relu[0][0]'] |  | | conv2\_block2\_2\_bn | (BatchNormal (None, 56, 56, 64) | 256 | ['conv2\_block2\_2\_conv[0][0]'] |  | | conv2\_block2\_2\_relu | (Activatio (None, 56, 56, 64) | 0 | ['conv2\_block2\_2\_bn[0][0]'] |  | | conv2\_block2\_3\_conv | (Conv2D) (None, 56, 56, 256) | 16640 | ['conv2\_block2\_2\_relu[0][0]'] |  | | conv2\_block2\_3\_bn | (BatchNormal (None, 56, 56, 256) | 1024 | ['conv2\_block2\_3\_conv[0][0]'] |  | | conv2\_block2\_add | (Add) (None, 56, 56, 256) | 0 | ['conv2\_block1\_out[0][0]', |  | | conv2\_block2\_out | (Activation) (None, 56, 56, 256) | 0 | ['conv2\_block2\_add[0][0]'] |  | | conv2\_block3\_1\_conv | (Conv2D) (None, 56, 56, 64) | 16448 | ['conv2\_block2\_out[0][0]'] |  | | conv2\_block3\_1\_bn | (BatchNormal (None, 56, 56, 64) | 256 | ['conv2\_block3\_1\_conv[0][0]'] |  | | conv2\_block3\_1\_relu | (Activatio (None, 56, 56, 64) | 0 | ['conv2\_block3\_1\_bn[0][0]'] |  | | conv2\_block3\_2\_conv | (Conv2D) (None, 56, 56, 64) | 36928 | ['conv2\_block3\_1\_relu[0][0]'] |  | | conv2\_block3\_2\_bn | (BatchNormal (None, 56, 56, 64) | 256 | ['conv2\_block3\_2\_conv[0][0]'] |  | | conv2\_block3\_2\_relu | (Activatio (None, 56, 56, 64) | 0 | ['conv2\_block3\_2\_bn[0][0]'] |  | | conv2\_block3\_3\_conv | (Conv2D) (None, 56, 56, 256) | 16640 | ['conv2\_block3\_2\_relu[0][0]'] |  | | conv2\_block3\_3\_bn | (BatchNormal (None, 56, 56, 256) | 1024 | ['conv2\_block3\_3\_conv[0][0]'] |  | | conv2\_block3\_add | (Add) (None, 56, 56, 256) | 0 | ['conv2\_block2\_out[0][0]', |  | | conv2\_block3\_out | (Activation) (None, 56, 56, 256) | 0 | ['conv2\_block3\_add[0][0]'] |  | | conv3\_block1\_1\_conv | (Conv2D) (None, 28, 28, 128) | 32896 | ['conv2\_block3\_out[0][0]'] |  | | conv3\_block1\_1\_bn | (BatchNormal (None, 28, 28, 128) | 512 | ['conv3\_block1\_1\_conv[0][0]'] |  | | conv3\_block1\_1\_relu | (Activatio (None, 28, 28, 128) | 0 | ['conv3\_block1\_1\_bn[0][0]'] |  | | conv3\_block1\_2\_conv | (Conv2D) (None, 28, 28, 128) | 147584 | ['conv3\_block1\_1\_relu[0][0]'] |  | | conv3\_block1\_2\_bn | (BatchNormal (None, 28, 28, 128) | 512 | ['conv3\_block1\_2\_conv[0][0]'] |  | | conv3\_block1\_2\_relu | (Activatio (None, 28, 28, 128) | 0 | ['conv3\_block1\_2\_bn[0][0]'] |  | | conv3\_block1\_0\_conv | (Conv2D) (None, 28, 28, 512) | 131584 | ['conv2\_block3\_out[0][0]'] |  | | conv3\_block1\_3\_conv | (Conv2D) (None, 28, 28, 512) | 66048 | ['conv3\_block1\_2\_relu[0][0]'] |  | | conv3\_block1\_0\_bn | (BatchNormal (None, 28, 28, 512) | 2048 | ['conv3\_block1\_0\_conv[0][0]'] |  | | conv3\_block1\_3\_bn | (BatchNormal (None, 28, 28, 512) | 2048 | ['conv3\_block1\_3\_conv[0][0]'] |  | | conv3\_block1\_add | (Add) (None, 28, 28, 512) | 0 | ['conv3\_block1\_0\_bn[0][0]', |  | | conv3\_block1\_out | (Activation) (None, 28, 28, 512) | 0 | ['conv3\_block1\_add[0][0]'] |  | | conv3\_block2\_1\_conv | (Conv2D) (None, 28, 28, 128) | 65664 | ['conv3\_block1\_out[0][0]'] |  | | conv3\_block2\_1\_bn | (BatchNormal (None, 28, 28, 128) | 512 | ['conv3\_block2\_1\_conv[0][0]'] |  | | conv3\_block2\_1\_relu | (Activatio (None, 28, 28, 128) | 0 | ['conv3\_block2\_1\_bn[0][0]'] |  | | conv3\_block2\_2\_conv | (Conv2D) (None, 28, 28, 128) | 147584 | ['conv3\_block2\_1\_relu[0][0]'] |  | | conv3\_block2\_2\_bn | (BatchNormal (None, 28, 28, 128) | 512 | ['conv3\_block2\_2\_conv[0][0]'] |  | | conv3\_block2\_2\_relu | (Activatio (None, 28, 28, 128) | 0 | ['conv3\_block2\_2\_bn[0][0]'] |  | | conv3\_block2\_3\_conv | (Conv2D) (None, 28, 28, 512) | 66048 | ['conv3\_block2\_2\_relu[0][0]'] |  | | conv3\_block2\_3\_bn | (BatchNormal (None, 28, 28, 512) | 2048 | ['conv3\_block2\_3\_conv[0][0]'] |  | | conv3\_block2\_add | (Add) (None, 28, 28, 512) | 0 | ['conv3\_block1\_out[0][0]', |  | | conv3\_block2\_out | (Activation) (None, 28, 28, 512) | 0 | ['conv3\_block2\_add[0][0]'] |  | | conv3\_block3\_1\_conv | (Conv2D) (None, 28, 28, 128) | 65664 | ['conv3\_block2\_out[0][0]'] |  | | conv3\_block3\_1\_bn | (BatchNormal (None, 28, 28, 128) | 512 | ['conv3\_block3\_1\_conv[0][0]'] |  | | conv3\_block3\_1\_relu | (Activatio (None, 28, 28, 128) | 0 | ['conv3\_block3\_1\_bn[0][0]'] |  | | conv3\_block3\_2\_conv | (Conv2D) (None, 28, 28, 128) | 147584 | ['conv3\_block3\_1\_relu[0][0]'] |  | | conv3\_block3\_2\_bn | (BatchNormal (None, 28, 28, 128) | 512 | ['conv3\_block3\_2\_conv[0][0]'] |  | | conv3\_block3\_2\_relu | (Activatio (None, 28, 28, 128) | 0 | ['conv3\_block3\_2\_bn[0][0]'] |  | | conv3\_block3\_3\_conv | (Conv2D) (None, 28, 28, 512) | 66048 | ['conv3\_block3\_2\_relu[0][0]'] |  | | conv3\_block3\_3\_bn | (BatchNormal (None, 28, 28, 512) | 2048 | ['conv3\_block3\_3\_conv[0][0]'] |  | | conv3\_block3\_add | (Add) (None, 28, 28, 512) | 0 | ['conv3\_block2\_out[0][0]', |  | | conv3\_block3\_out | (Activation) (None, 28, 28, 512) | 0 | ['conv3\_block3\_add[0][0]'] |  | | conv3\_block4\_1\_conv | (Conv2D) (None, 28, 28, 128) | 65664 | ['conv3\_block3\_out[0][0]'] |  | | conv3\_block4\_1\_bn | (BatchNormal (None, 28, 28, 128) | 512 | ['conv3\_block4\_1\_conv[0][0]'] |  | | conv3\_block4\_1\_relu | (Activatio (None, 28, 28, 128) | 0 | ['conv3\_block4\_1\_bn[0][0]'] |  | | conv3\_block4\_2\_conv | (Conv2D) (None, 28, 28, 128) | 147584 | ['conv3\_block4\_1\_relu[0][0]'] |  | | conv3\_block4\_2\_bn | (BatchNormal (None, 28, 28, 128) | 512 | ['conv3\_block4\_2\_conv[0][0]'] |  | | conv3\_block4\_2\_relu | (Activatio (None, 28, 28, 128) | 0 | ['conv3\_block4\_2\_bn[0][0]'] |  | | conv3\_block4\_3\_conv | (Conv2D) (None, 28, 28, 512) | 66048 | ['conv3\_block4\_2\_relu[0][0]'] |  | | conv3\_block4\_3\_bn | (BatchNormal (None, 28, 28, 512) | 2048 | ['conv3\_block4\_3\_conv[0][0]'] |  | | conv3\_block4\_add | (Add) (None, 28, 28, 512) | 0 | ['conv3\_block3\_out[0][0]', |  | | conv3\_block4\_out | (Activation) (None, 28, 28, 512) | 0 | ['conv3\_block4\_add[0][0]'] |  | | conv4\_block1\_1\_conv | (Conv2D) (None, 14, 14, 256) | 131328 | ['conv3\_block4\_out[0][0]'] |  | | conv4\_block1\_1\_bn | (BatchNormal (None, 14, 14, 256) | 1024 | ['conv4\_block1\_1\_conv[0][0]'] |  | | conv4\_block1\_1\_relu | (Activatio (None, 14, 14, 256) | 0 | ['conv4\_block1\_1\_bn[0][0]'] |  | | conv4\_block1\_2\_conv | (Conv2D) (None, 14, 14, 256) | 590080 | ['conv4\_block1\_1\_relu[0][0]'] |  | | conv4\_block1\_2\_bn | (BatchNormal (None, 14, 14, 256) | 1024 | ['conv4\_block1\_2\_conv[0][0]'] |  | | conv4\_block1\_2\_relu | (Activatio (None, 14, 14, 256) | 0 | ['conv4\_block1\_2\_bn[0][0]'] |  | | conv4\_block1\_0\_conv | (Conv2D) (None, 14, 14, 1024 | 525312 | ['conv3\_block4\_out[0][0]'] |  | | conv4\_block1\_3\_conv | (Conv2D) (None, 14, 14, 1024 | 263168 | ['conv4\_block1\_2\_relu[0][0]'] |  | | conv4\_block1\_0\_bn | (BatchNormal (None, 14, 14, 1024 | 4096 | ['conv4\_block1\_0\_conv[0][0]'] |  | | conv4\_block1\_3\_bn | (BatchNormal (None, 14, 14, 1024 | 4096 | ['conv4\_block1\_3\_conv[0][0]'] |  | | conv4\_block1\_add | (Add) (None, 14, 14, 1024 | 0 | ['conv4\_block1\_0\_bn[0][0]', |  | | conv4\_block1\_out | (Activation) (None, 14, 14, 1024 | 0 | ['conv4\_block1\_add[0][0]'] |  | | conv4\_block2\_1\_conv | (Conv2D) (None, 14, 14, 256) | 262400 | ['conv4\_block1\_out[0][0]'] |  | | conv4\_block2\_1\_bn | (BatchNormal (None, 14, 14, 256) | 1024 | ['conv4\_block2\_1\_conv[0][0]'] |  | | conv4\_block2\_1\_relu | (Activatio (None, 14, 14, 256) | 0 | ['conv4\_block2\_1\_bn[0][0]'] |  | | conv4\_block2\_2\_conv | (Conv2D) (None, 14, 14, 256) | 590080 | ['conv4\_block2\_1\_relu[0][0]'] |  | | conv4\_block2\_2\_bn | (BatchNormal (None, 14, 14, 256) | 1024 | ['conv4\_block2\_2\_conv[0][0]'] |  | | conv4\_block2\_2\_relu | (Activatio (None, 14, 14, 256) | 0 | ['conv4\_block2\_2\_bn[0][0]'] |  | | conv4\_block2\_3\_conv | (Conv2D) (None, 14, 14, 1024 | 263168 | ['conv4\_block2\_2\_relu[0][0]'] |  | | conv4\_block2\_3\_bn | (BatchNormal (None, 14, 14, 1024 | 4096 | ['conv4\_block2\_3\_conv[0][0]'] |  | | conv4\_block2\_add | (Add) (None, 14, 14, 1024 | 0 | ['conv4\_block1\_out[0][0]', |  | | conv4\_block2\_out | (Activation) (None, 14, 14, 1024 | 0 | ['conv4\_block2\_add[0][0]'] |  | | conv4\_block3\_1\_conv | (Conv2D) (None, 14, 14, 256) | 262400 | ['conv4\_block2\_out[0][0]'] |  | | conv4\_block3\_1\_bn | (BatchNormal (None, 14, 14, 256) | 1024 | ['conv4\_block3\_1\_conv[0][0]'] |  | | conv4\_block3\_1\_relu | (Activatio (None, 14, 14, 256) | 0 | ['conv4\_block3\_1\_bn[0][0]'] |  | | conv4\_block3\_2\_conv | (Conv2D) (None, 14, 14, 256) | 590080 | ['conv4\_block3\_1\_relu[0][0]'] |  | | conv4\_block3\_2\_bn | (BatchNormal (None, 14, 14, 256) | 1024 | ['conv4\_block3\_2\_conv[0][0]'] |  | | conv4\_block3\_2\_relu | (Activatio (None, 14, 14, 256) | 0 | ['conv4\_block3\_2\_bn[0][0]'] |  | | conv4\_block3\_3\_conv | (Conv2D) (None, 14, 14, 1024 | 263168 | ['conv4\_block3\_2\_relu[0][0]'] |  | | conv4\_block3\_3\_bn | (BatchNormal (None, 14, 14, 1024 | 4096 | ['conv4\_block3\_3\_conv[0][0]'] |  | | conv4\_block3\_add | (Add) (None, 14, 14, 1024 | 0 | ['conv4\_block2\_out[0][0]', |  | | conv4\_block3\_out | (Activation) (None, 14, 14, 1024 | 0 | ['conv4\_block3\_add[0][0]'] |  | | conv4\_block4\_1\_conv | (Conv2D) (None, 14, 14, 256) | 262400 | ['conv4\_block3\_out[0][0]'] |  | | conv4\_block4\_1\_bn | (BatchNormal (None, 14, 14, 256) | 1024 | ['conv4\_block4\_1\_conv[0][0]'] |  | | conv4\_block4\_1\_relu | (Activatio (None, 14, 14, 256) | 0 | ['conv4\_block4\_1\_bn[0][0]'] |  | | conv4\_block4\_2\_conv | (Conv2D) (None, 14, 14, 256) | 590080 | ['conv4\_block4\_1\_relu[0][0]'] |  | | conv4\_block4\_2\_bn | (BatchNormal (None, 14, 14, 256) | 1024 | ['conv4\_block4\_2\_conv[0][0]'] |  | | conv4\_block4\_2\_relu | (Activatio (None, 14, 14, 256) | 0 | ['conv4\_block4\_2\_bn[0][0]'] |  | | conv4\_block4\_3\_conv | (Conv2D) (None, 14, 14, 1024 | 263168 | ['conv4\_block4\_2\_relu[0][0]'] |  | | conv4\_block4\_3\_bn | (BatchNormal (None, 14, 14, 1024 | 4096 | ['conv4\_block4\_3\_conv[0][0]'] |  | | conv4\_block4\_add | (Add) (None, 14, 14, 1024 | 0 | ['conv4\_block3\_out[0][0]', |  | | conv4\_block4\_out | (Activation) (None, 14, 14, 1024 | 0 | ['conv4\_block4\_add[0][0]'] |  | | conv4\_block5\_1\_conv | (Conv2D) (None, 14, 14, 256) | 262400 | ['conv4\_block4\_out[0][0]'] |  | | conv4\_block5\_1\_bn | (BatchNormal (None, 14, 14, 256) | 1024 | ['conv4\_block5\_1\_conv[0][0]'] |  | | conv4\_block5\_1\_relu | (Activatio (None, 14, 14, 256) | 0 | ['conv4\_block5\_1\_bn[0][0]'] |  | | conv4\_block5\_2\_conv | (Conv2D) (None, 14, 14, 256) | 590080 | ['conv4\_block5\_1\_relu[0][0]'] |  | | conv4\_block5\_2\_bn | (BatchNormal (None, 14, 14, 256) | 1024 | ['conv4\_block5\_2\_conv[0][0]'] |  | | conv4\_block5\_2\_relu | (Activatio (None, 14, 14, 256) | 0 | ['conv4\_block5\_2\_bn[0][0]'] |  | | conv4\_block5\_3\_conv | (Conv2D) (None, 14, 14, 1024 | 263168 | ['conv4\_block5\_2\_relu[0][0]'] |  | | conv4\_block5\_3\_bn | (BatchNormal (None, 14, 14, 1024 | 4096 | ['conv4\_block5\_3\_conv[0][0]'] |  | | conv4\_block5\_add | (Add) (None, 14, 14, 1024 | 0 | ['conv4\_block4\_out[0][0]', |  | | conv4\_block5\_out | (Activation) (None, 14, 14, 1024 | 0 | ['conv4\_block5\_add[0][0]'] |  | | conv4\_block6\_1\_conv | (Conv2D) (None, 14, 14, 256) | 262400 | ['conv4\_block5\_out[0][0]'] |  | | conv4\_block6\_1\_bn | (BatchNormal (None, 14, 14, 256) | 1024 | ['conv4\_block6\_1\_conv[0][0]'] |  | | conv4\_block6\_1\_relu | (Activatio (None, 14, 14, 256) | 0 | ['conv4\_block6\_1\_bn[0][0]'] |  | | conv4\_block6\_2\_conv | (Conv2D) (None, 14, 14, 256) | 590080 | ['conv4\_block6\_1\_relu[0][0]'] |  | | conv4\_block6\_2\_bn | (BatchNormal (None, 14, 14, 256) | 1024 | ['conv4\_block6\_2\_conv[0][0]'] |  | | conv4\_block6\_2\_relu | (Activatio (None, 14, 14, 256) | 0 | ['conv4\_block6\_2\_bn[0][0]'] |  | | conv4\_block6\_3\_conv | (Conv2D) (None, 14, 14, 1024 | 263168 | ['conv4\_block6\_2\_relu[0][0]'] |  | | conv4\_block6\_3\_bn | (BatchNormal (None, 14, 14, 1024 | 4096 | ['conv4\_block6\_3\_conv[0][0]'] |  | | conv4\_block6\_add | (Add) (None, 14, 14, 1024 | 0 | ['conv4\_block5\_out[0][0]', |  | | conv4\_block6\_out | (Activation) (None, 14, 14, 1024 | 0 | ['conv4\_block6\_add[0][0]'] |  | | conv5\_block1\_1\_conv | (Conv2D) (None, 7, 7, 512) | 524800 | ['conv4\_block6\_out[0][0]'] |  | | conv5\_block1\_1\_bn | (BatchNormal (None, 7, 7, 512) | 2048 | ['conv5\_block1\_1\_conv[0][0]'] |  | | conv5\_block1\_1\_relu | (Activatio (None, 7, 7, 512) | 0 | ['conv5\_block1\_1\_bn[0][0]'] |  | | conv5\_block1\_2\_conv | (Conv2D) (None, 7, 7, 512) | 2359808 | ['conv5\_block1\_1\_relu[0][0]'] |  | | conv5\_block1\_2\_bn | (BatchNormal (None, 7, 7, 512) | 2048 | ['conv5\_block1\_2\_conv[0][0]'] |  | | conv5\_block1\_2\_relu | (Activatio (None, 7, 7, 512) | 0 | ['conv5\_block1\_2\_bn[0][0]'] |  | | conv5\_block1\_0\_conv | (Conv2D) (None, 7, 7, 2048) | 2099200 | ['conv4\_block6\_out[0][0]'] |  | | conv5\_block1\_3\_conv | (Conv2D) (None, 7, 7, 2048) | 1050624 | ['conv5\_block1\_2\_relu[0][0]'] |  | | conv5\_block1\_0\_bn | (BatchNormal (None, 7, 7, 2048) | 8192 | ['conv5\_block1\_0\_conv[0][0]'] |  | | conv5\_block1\_3\_bn | (BatchNormal (None, 7, 7, 2048) | 8192 | ['conv5\_block1\_3\_conv[0][0]'] |  | | conv5\_block1\_add | (Add) (None, 7, 7, 2048) | 0 | ['conv5\_block1\_0\_bn[0][0]', |  | | conv5\_block1\_out | (Activation) (None, 7, 7, 2048) | 0 | ['conv5\_block1\_add[0][0]'] |  | | conv5\_block2\_1\_conv | (Conv2D) (None, 7, 7, 512) | 1049088 | ['conv5\_block1\_out[0][0]'] |  | | conv5\_block2\_1\_bn | (BatchNormal (None, 7, 7, 512) | 2048 | ['conv5\_block2\_1\_conv[0][0]'] |  | | conv5\_block2\_1\_relu | (Activatio (None, 7, 7, 512) | 0 | ['conv5\_block2\_1\_bn[0][0]'] |  | | conv5\_block2\_2\_conv | (Conv2D) (None, 7, 7, 512) | 2359808 | ['conv5\_block2\_1\_relu[0][0]'] |  | | conv5\_block2\_2\_bn | (BatchNormal (None, 7, 7, 512) | 2048 | ['conv5\_block2\_2\_conv[0][0]'] |  | | conv5\_block2\_2\_relu | (Activatio (None, 7, 7, 512) | 0 | ['conv5\_block2\_2\_bn[0][0]'] |  | | conv5\_block2\_3\_conv | (Conv2D) (None, 7, 7, 2048) | 1050624 | ['conv5\_block2\_2\_relu[0][0]'] |  | | conv5\_block2\_3\_bn | (BatchNormal (None, 7, 7, 2048) | 8192 | ['conv5\_block2\_3\_conv[0][0]'] |  | | conv5\_block2\_add | (Add) (None, 7, 7, 2048) | 0 | ['conv5\_block1\_out[0][0]', |  | | conv5\_block2\_out | (Activation) (None, 7, 7, 2048) | 0 | ['conv5\_block2\_add[0][0]'] |  | | conv5\_block3\_1\_conv | (Conv2D) (None, 7, 7, 512) | 1049088 | ['conv5\_block2\_out[0][0]'] |  | | conv5\_block3\_1\_bn | (BatchNormal (None, 7, 7, 512) | 2048 | ['conv5\_block3\_1\_conv[0][0]'] |  | | conv5\_block3\_1\_relu | (Activatio (None, 7, 7, 512) | 0 | ['conv5\_block3\_1\_bn[0][0]'] |  | | conv5\_block3\_2\_conv | (Conv2D) (None, 7, 7, 512) | 2359808 | ['conv5\_block3\_1\_relu[0][0]'] |  | | conv5\_block3\_2\_bn | (BatchNormal (None, 7, 7, 512) | 2048 | ['conv5\_block3\_2\_conv[0][0]'] |  | | conv5\_block3\_2\_relu | (Activatio (None, 7, 7, 512) | 0 | ['conv5\_block3\_2\_bn[0][0]'] |  | | conv5\_block3\_3\_conv | (Conv2D) (None, 7, 7, 2048) | 1050624 | ['conv5\_block3\_2\_relu[0][0]'] |  | | conv5\_block3\_3\_bn | (BatchNormal (None, 7, 7, 2048) | 8192 | ['conv5\_block3\_3\_conv[0][0]'] |  | | conv5\_block3\_add | (Add) (None, 7, 7, 2048) | 0 | ['conv5\_block2\_out[0][0]', |  | | conv5\_block3\_out | (Activation) (None, 7, 7, 2048) | 0 | ['conv5\_block3\_add[0][0]'] |  | | avg\_pool | (GlobalAveragePooling (None, 2048) | 0 | ['conv5\_block3\_out[0][0]'] |  | | predictions | (Dense) (None, 1000) | 2049000 | ['avg\_pool[0][0]'] |  | |
| **VGG16 Summary** |
| |  |  |  |  | | --- | --- | --- | --- | | **Layer (type)** | **Output Shape** | **Parameters** | **Connected to** | |  | | input\_4 | (InputLayer) [(None, 224, 224, | 3)] | 0 |  | | block1\_conv1 | (Conv2D) (None, 224, 224, | 64) | 1792 |  | | block1\_conv2 | (Conv2D) (None, 224, 224, | 64) | 36928 |  | | block1\_pool | (MaxPooling2D) (None, 112, 112, | 64) | 0 |  | | block2\_conv1 | (Conv2D) (None, 112, 112, | 128) | 73856 |  | | block2\_conv2 | (Conv2D) (None, 112, 112, | 128) | 147584 |  | | block2\_pool | (MaxPooling2D) (None, 56, 56, | 128) | 0 |  | | block3\_conv1 | (Conv2D) (None, 56, 56, | 256) | 295168 |  | | block3\_conv2 | (Conv2D) (None, 56, 56, | 256) | 590080 |  | | block3\_conv3 | (Conv2D) (None, 56, 56, | 256) | 590080 |  | | block3\_pool | (MaxPooling2D) (None, 28, 28, | 256) | 0 |  | | block4\_conv1 | (Conv2D) (None, 28, 28, | 512) | 1180160 |  | | block4\_conv2 | (Conv2D) (None, 28, 28, | 512) | 2359808 |  | | block4\_conv3 | (Conv2D) (None, 28, 28, | 512) | 2359808 |  | | block4\_pool | (MaxPooling2D) (None, 14, 14, | 512) | 0 |  | | block5\_conv1 | (Conv2D) (None, 14, 14, | 512) | 2359808 |  | | block5\_conv2 | (Conv2D) (None, 14, 14, | 512) | 2359808 |  | | block5\_conv3 | (Conv2D) (None, 14, 14, | 512) | 2359808 |  | | block5\_pool | (MaxPooling2D) (None, 7, 7, | 512) | 0 |  | | flatten | (Flatten) (None, | 25088) | 0 |  | | fc1 | (Dense) (None, | 4096) | 102764544 |  | | fc2 | (Dense) (None, | 4096) | 16781312 |  | | predictions | (Dense) (None, | 1000) | 4097000 |  | |