Facial Emotion Detection

Capstone Project MIT-PE ADSP

Albert Solana

Eight Key TakeAways

1. **Project Aim**: Developed a model to **identify human emotions from facial expressions** in images.
2. **Data Utilization**: Used a dataset of **grayscale images** labeled with **four emotions**—happy, neutral, sad, surprise—for model training.
3. **Model Exploration**: Tested various **neural network architectures**, including custom CNNs and transfer learning models like VGG16, ResNet50V2, and EfficientNetV2B0.
4. **Performance Metrics**: **Evaluated models** based on **accuracy**, precision, recall, and F1-score with the help of confusion matrices.
5. **Best Performer**: A **complex CNN model** with five convolutional blocks demonstrated superior performance.
6. **Ethical Considerations**: Emphasized the importance of **privacy and fairness** in deploying the emotion detection model in real-world scenarios.
7. **Real-World Application**: Potential uses in enhancing user interface experiences and **supporting mental health professionals**.
8. **Future Work**: Suggestions for **model improvement** and additional **ethical safeguards**.

Overview of the Problem

Emotion Recognition:

The need to understand **human emotions** accurately using technology.

Application Breadth:

Relevance in sectors like customer service, healthcare, and education.

Current Gaps:

Limitations of existing solutions in accuracy and real-world usability.

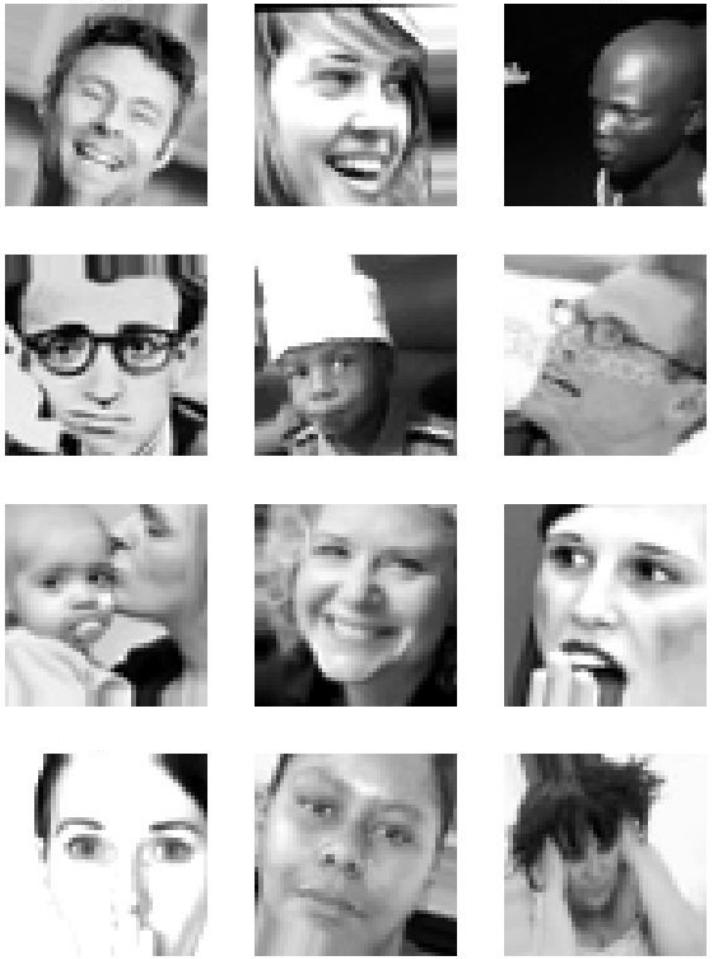


Fig 1: Several random images from the dataset.

Overview of the Problem

Emotion Recognition:

The need to understand **human emotions** accurately using technology.

Application Breadth:

Relevance in sectors like customer service, healthcare, and education.

Current Gaps:

Limitations of existing solutions in accuracy and real-world usability.

Fig 2: Several random images from the dataset with a possible human classification.

Solution Approach

- **Dataset**: Utilization of a **curated dataset of facial expressions** with a process of **data augmentation.**
- **Process**: **Training**, **validation**, and **testing** phases.
- **Model Selection**: Exploration of **Convolutional Neural Networks** and **transfer learning models**
- **Model evaluation**: Using **accuracy** as the **key metric** to benchmark models.

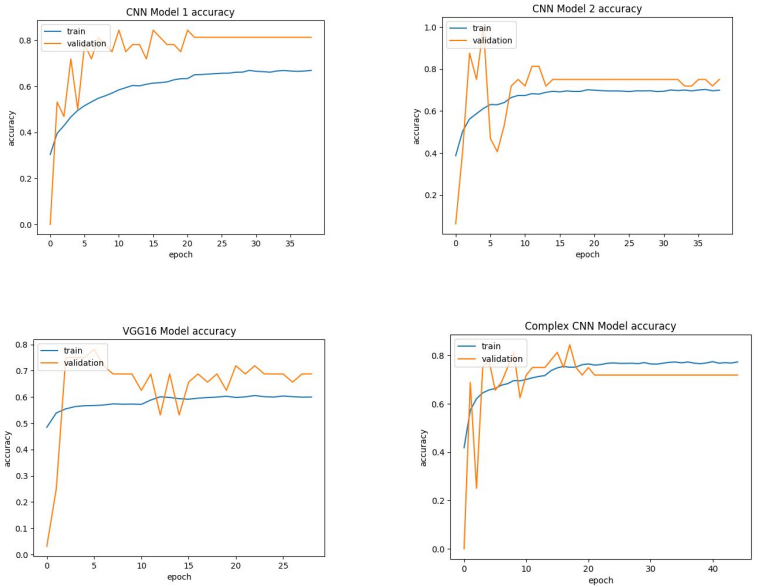


Fig 3: Model evaluation: training and validation plots among several tested models.

Key Findings & Insights

- The **custom complex CNN model** is proposed for adoption as the final solution.
- Models show varying degrees of success, with some like the complex CNN achieving a high level of accuracy (over 80%), suggesting that **deeper architectures can capture better the features in facial expressions**.
- Across models, **'happy' and 'surprise' emotions are generally identified with high accuracy**, implying that these emotions have distinct features that are easily captured by the CNN layers.
- **'Neutral' and 'sad' emotions are more challenging** for the models to distinguish, often being confused with one another.
- **Transfer learning models did not outperform the custom complex CNN**, for this specific dataset and problem, custom-designed architectures can be more effective than off-the-shelf pre-trained models.

Model	Loss	Accuracy
CNN Model 1	0.6502	0.7031
CNN Model 2	0.5752	0.7656
VGG16 Model	0.8578	0.6406
ResNet50V2 Model	0.9988	0.6172
EfficientNet Model	0.9597	0.6328
Complex CNN Model	0.5056	0.8125

Table 1: Comparative Performance of Different CNN Architectures and Transfer Learning Models

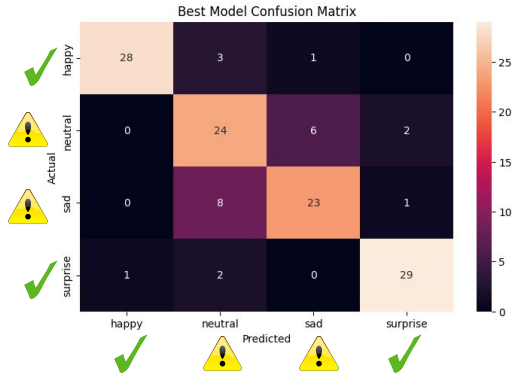


Fig 4: Best Model Confusion matrix in which 'happy' and 'surprise' emotions are well identified, whereas 'neutral' and 'sad' still are more challenging.

Risks & Challenges

- **Data Quality**: The project outcome depend deeply in the dataset quality.
- **Overfitting Risk**: We had to create models not only adapted to the training dataset. We want models to **perform** excellent in **production environments**.
- **Real-World Viability**: Challenges in applying the solution in a real-world setting.
- To be really careful about the **ethical considerations** surrounding its deployment, such as **privacy concerns** and potential **biases**.

Recommendations and Next Steps

- **Data Augmentation**: Testing **more sophisticated data augmentation strategies** might help the model generalize better, particularly for underperforming classes.
- **Increased Data**: More training data, particularly for the **underrepresented and challenging emotions**, could improve model learning.
- **Hyperparameter Tuning**: Continue tuning the models hyperparameters could optimize model performance.
- **Model creation**: Continue creating **new custom models** to outperform the ones generated.
- **Real world application**: Deploy the model into a real world scenario to adjust accordingly to the use case to solve.

Thank You

Albert Solana

Appendix

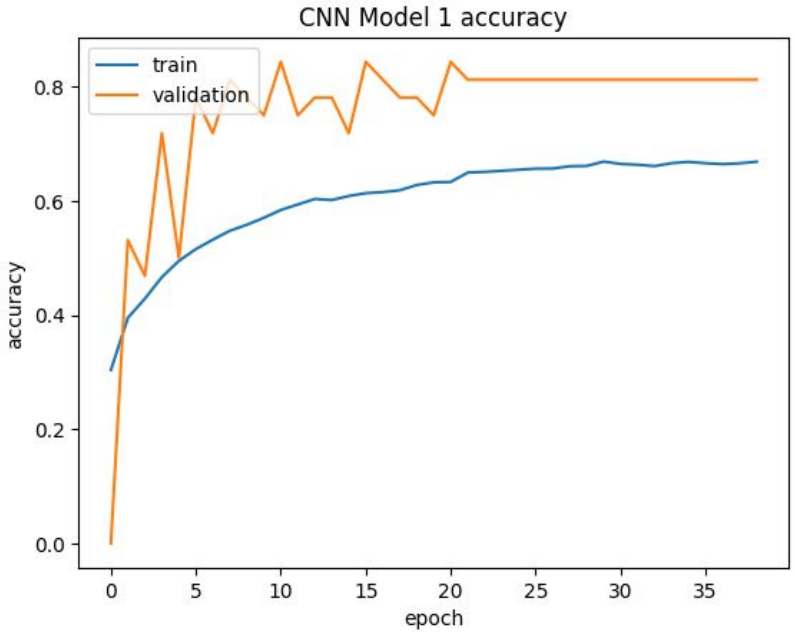
Models Comparison: CNN Model 1

Architecture

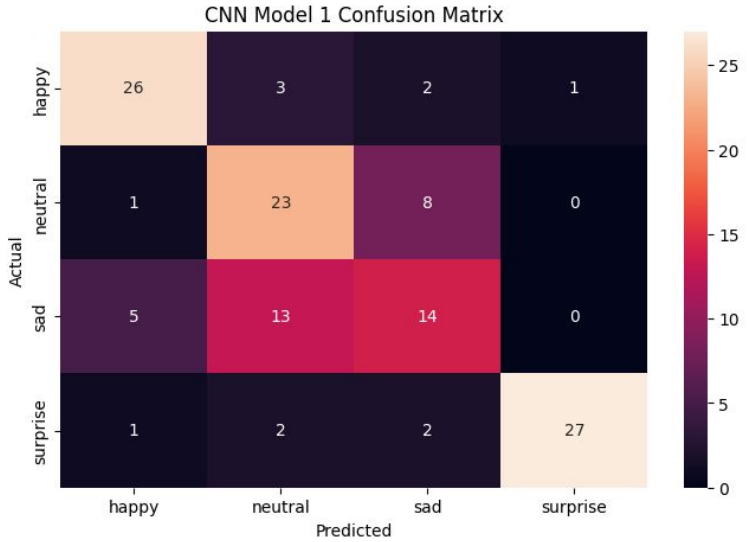
```
# Intializing a sequential model
model_1 = Sequential()

model_1.add(Conv2D(64, (3, 3), activation="relu", padding="same", input_shape=(img_width, img_height, color_layers)))
model_1.add(MaxPooling2D((2, 2), padding="same"))
model_1.add(Dropout(0.2))

# Adding second conv layer with 32 filters
model_1.add(Conv2D(32, (3, 3), activation="relu", padding="same"))
model_1.add(MaxPooling2D((2, 2), padding="same"))
model_1.add(Dropout(0.2))

# Add third conv layer with 32 filters and kernel size 3x3, padding 'same' followed by a Maxpooling2D layer
model_1.add(Conv2D(32, (3, 3), activation="relu", padding="same"))
model_1.add(MaxPooling2D((2, 2), padding="same"))
model_1.add(Dropout(0.2))

# Flattening the output of the conv layer after max pooling to make it ready for creating dense connections
model_1.add(Flatten())

# Adding a fully connected dense layers
model_1.add(Dense(512, activation="relu"))
model_1.add(Dense(64, activation="relu"))
model_1.add(Dense(4, activation="softmax"))
```

Models Comparison: CNN Model 1

Results



	precision	recall	f1-score	support
happy	0.79	0.81	0.80	32
neutral	0.56	0.72	0.63	32
sad	0.54	0.44	0.48	32
surprise	0.96	0.84	0.90	32
accuracy			0.70	128
macro avg	0.71	0.70	0.70	128
weighted avg	0.71	0.70	0.70	128



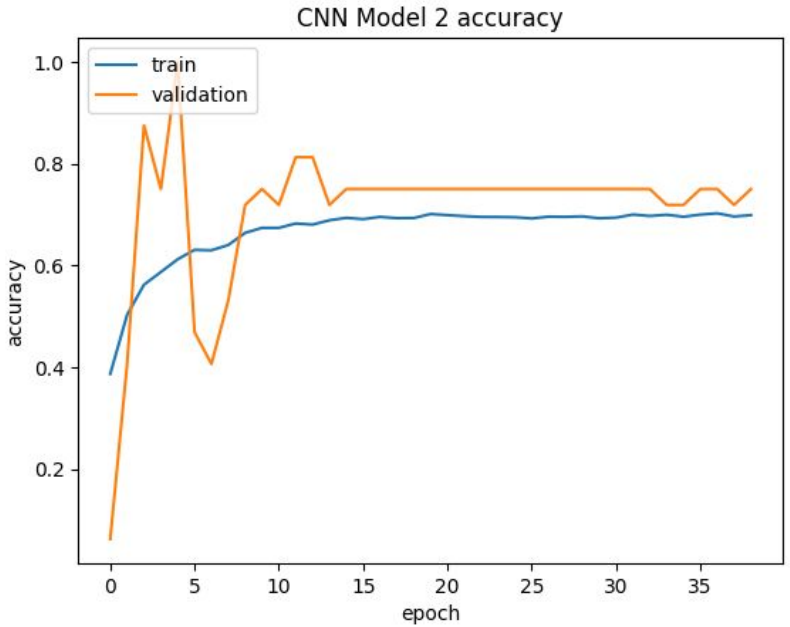
Models Comparison: CNN Model 2

Architecture

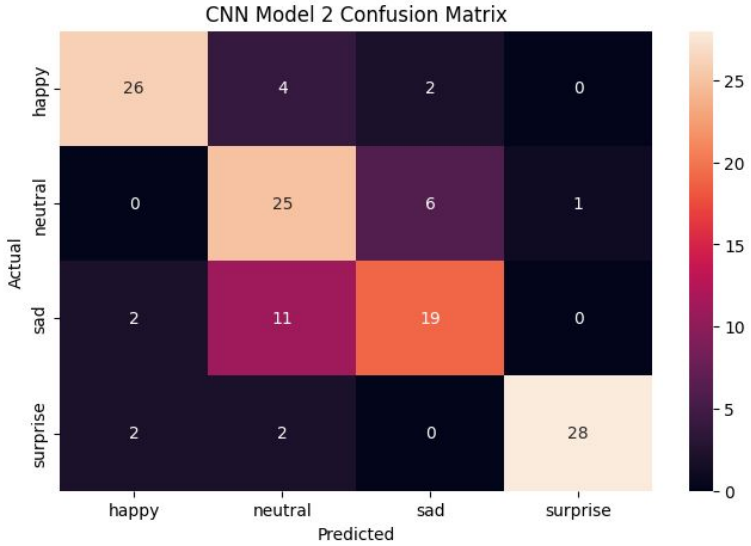
```
# Initializing a sequential model
model_2 = Sequential()

# First Convolutional Block
model_2.add(Conv2D(256, kernel_size=2, padding="same", input_shape=(48, 48, 1)))
model_2.add(BatchNormalization())
model_2.add(LeakyReLU(alpha=0.1))
model_2.add(MaxPooling2D(pool_size=2))

# Second Convolutional Block
model_2.add(Conv2D(128, kernel_size=2, padding="same"))
model_2.add(BatchNormalization())
model_2.add(LeakyReLU(alpha=0.1))
model_2.add(MaxPooling2D(pool_size=2))

# Third Convolutional Block
model_2.add(Conv2D(64, kernel_size=2, padding="same"))
model_2.add(BatchNormalization())
model_2.add(LeakyReLU(alpha=0.1))
model_2.add(MaxPooling2D(pool_size=2))

# Fourth Convolutional Block
model_2.add(Conv2D(32, kernel_size=2, padding="same"))
model_2.add(BatchNormalization())
model_2.add(LeakyReLU(alpha=0.1))
model_2.add(MaxPooling2D(pool_size=2))

# Flatten the output of the conv layers to feed into the dense layers
model_2.add(Flatten())

# Fully connected layers
model_2.add(Dense(512, activation="relu"))
model_2.add(Dense(128, activation="relu"))
model_2.add(Dense(64))
model_2.add(BatchNormalization())
model_2.add(ReLU())  # Using ReLU after batch normalization
model_2.add(Dense(4, activation="softmax"))
```

Models Comparison: CNN Model 2

Results



	precision	recall	f1-score	support
happy	0.87	0.81	0.84	32
neutral	0.60	0.78	0.68	32
sad	0.70	0.59	0.64	32
surprise	0.97	0.88	0.92	32
accuracy			0.77	128
macro avg	0.78	0.77	0.77	128
weighted avg	0.78	0.77	0.77	128



Models Comparison: VGG16 Model

Architecture

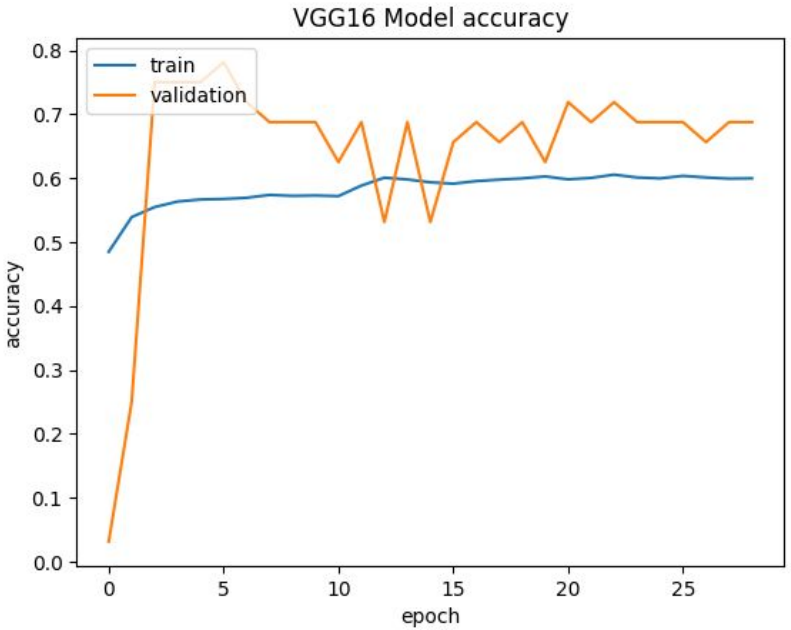
```
vgg_model = VGG16(weights="imagenet", include_top=False, input_shape=(img_width, img_height, color_layers))
vgg_model.summary()
```

```
# Define a new model that cuts VGG16 at the 'block3_pool' layer
model_output = vgg_model.get_layer("block3_pool").output
cut_model = Model(inputs=vgg_model.input, outputs=model_output)

for layer in vgg_model.layers:
    layer.trainable = False
```

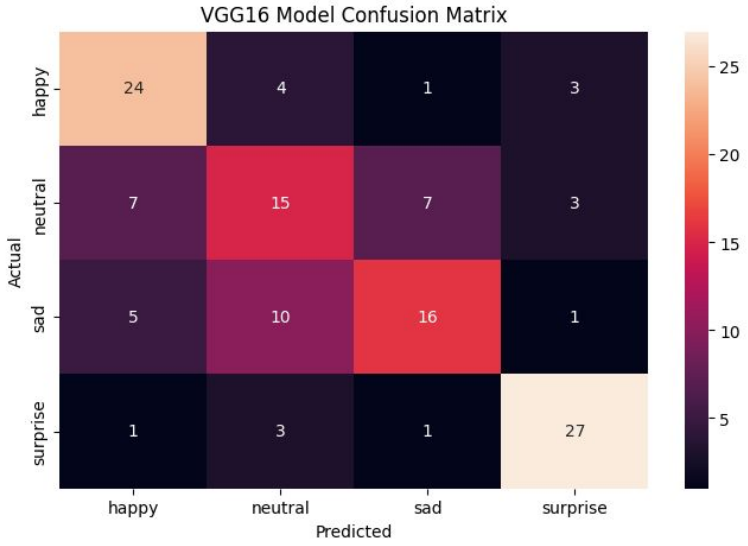
```
new_vgg16_model = Sequential()

# Adding the convolutional part of the VGG16 model from above until the cut layer
new_vgg16_model.add(cut_model)

# Reduces each feature map to a single value by averaging all elements
new_vgg16_model.add(GlobalAveragePooling2D())

# Adding full connected layers
new_vgg16_model.add(Dense(512, activation="relu"))
new_vgg16_model.add(Dense(128, activation="relu"))
new_vgg16_model.add(Dense(64))
new_vgg16_model.add(BatchNormalization())
new_vgg16_model.add(ReLU())  # Using ReLU after batch normalization

# Adding output layer
new_vgg16_model.add(Dense(4, activation="softmax"))
```

Models Comparison: VGG16 Model

Results



	precision	recall	f1-score	support
happy	0.65	0.75	0.70	32
neutral	0.47	0.47	0.47	32
sad	0.64	0.50	0.56	32
surprise	0.79	0.84	0.82	32
accuracy			0.64	128
macro avg	0.64	0.64	0.64	128
weighted avg	0.64	0.64	0.64	128



Models Comparison: ResNet V2 Model

Architecture

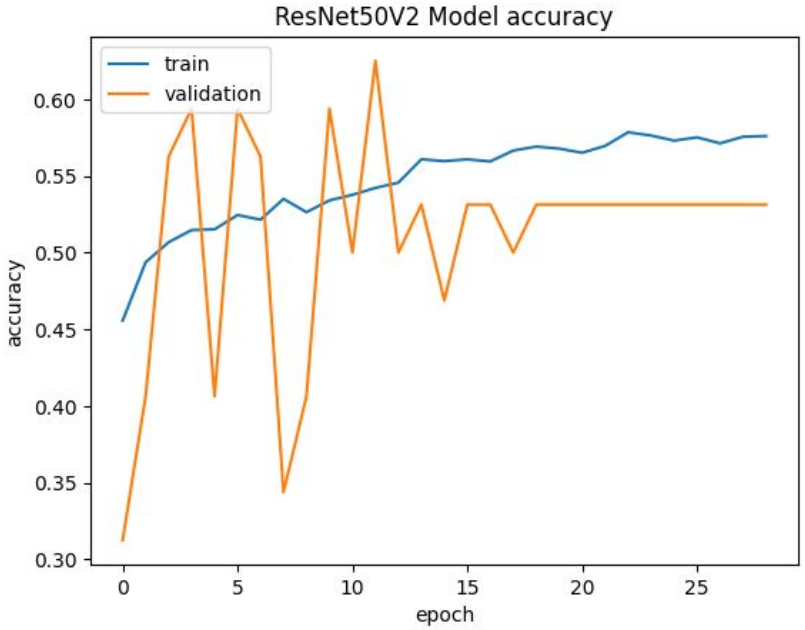
```
resnet_model = ResNet50V2(weights="imagenet", include_top=False, input_shape=(img_width, img_height, color_layers))
resnet_model.summary()
```

```
# Freezing the layers
for layer in resnet_model.layers:
    layer.trainable = False
```

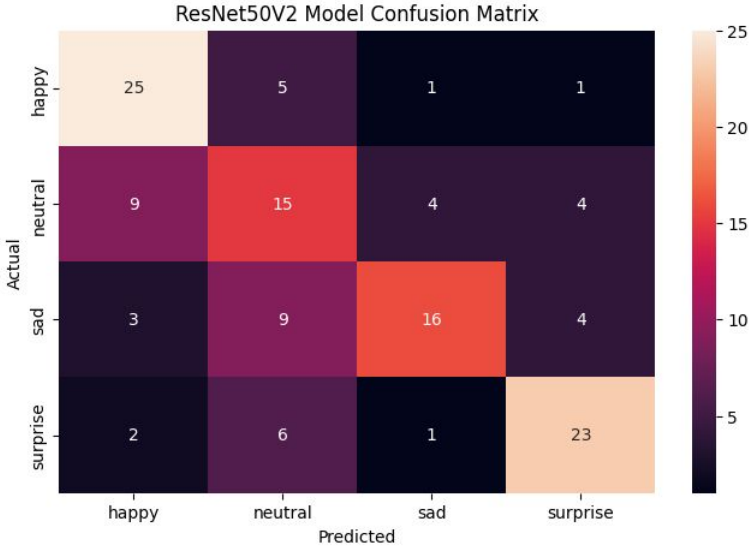
```
new_resnet_model = Sequential()
new_resnet_model.add(resnet_model)

# Reduces each feature map to a single value by averaging all elements
new_resnet_model.add(GlobalAveragePooling2D())

# Adding full connected layers
new_resnet_model.add(Dense(512, activation="relu"))
new_resnet_model.add(Dense(128, activation="relu"))
new_resnet_model.add(Dense(64))
new_resnet_model.add(BatchNormalization())
new_resnet_model.add(ReLU())  # Using ReLU after batch normalization

# Adding output layer
new_resnet_model.add(Dense(4, activation="softmax"))
```

Models Comparison: ResNet V2 Model

Results



	precision	recall	f1-score	support
happy	0.64	0.78	0.70	32
neutral	0.43	0.47	0.45	32
sad	0.73	0.50	0.59	32
surprise	0.72	0.72	0.72	32
accuracy			0.62	128
macro avg	0.63	0.62	0.62	128
weighted avg	0.63	0.62	0.62	128



Models Comparison: EfficientNet Model

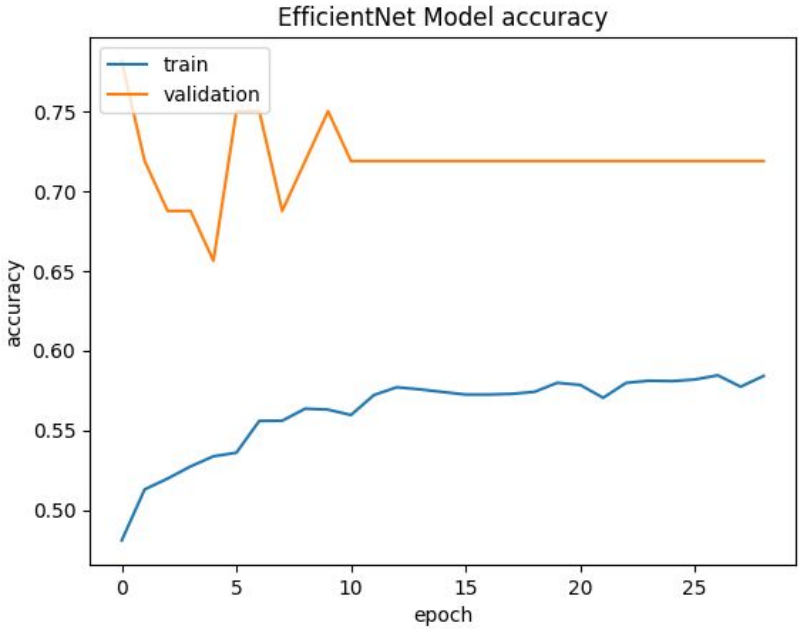
Architecture

```
efficient_model = EfficientNetV2B0(
    weights="imagenet", include_top=False, input_shape=(img_width, img_height, color_layers)
)
# Making all the layers of the efficient_model model non-trainable. i.e. freezing them
for layer in efficient_model.layers:
    layer.trainable = False

efficient_model.summary()
```

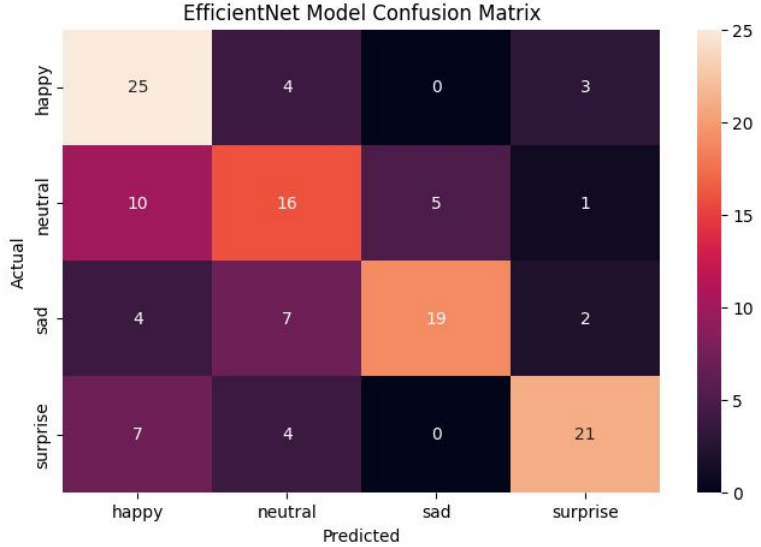
```
new_efficient_model = Sequential()
new_efficient_model.add(efficient_model)

# Reduces each feature map to a single value by averaging all elements
new_efficient_model.add(GlobalAveragePooling2D())

# Adding full connected layers
new_efficient_model.add(Dense(512, activation="relu"))
new_efficient_model.add(Dense(128, activation="relu"))
new_efficient_model.add(Dense(64))
new_efficient_model.add(BatchNormalization())
new_efficient_model.add(ReLU())  # Using ReLU after batch normalization

# Adding output layer
new_efficient_model.add(Dense(4, activation="softmax"))
```

Models Comparison: EfficientNet Model

Results



	precision	recall	f1-score	support
happy	0.54	0.78	0.64	32
neutral	0.52	0.50	0.51	32
sad	0.79	0.59	0.68	32
surprise	0.78	0.66	0.71	32
accuracy			0.63	128
macro avg	0.66	0.63	0.63	128
weighted avg	0.66	0.63	0.63	128



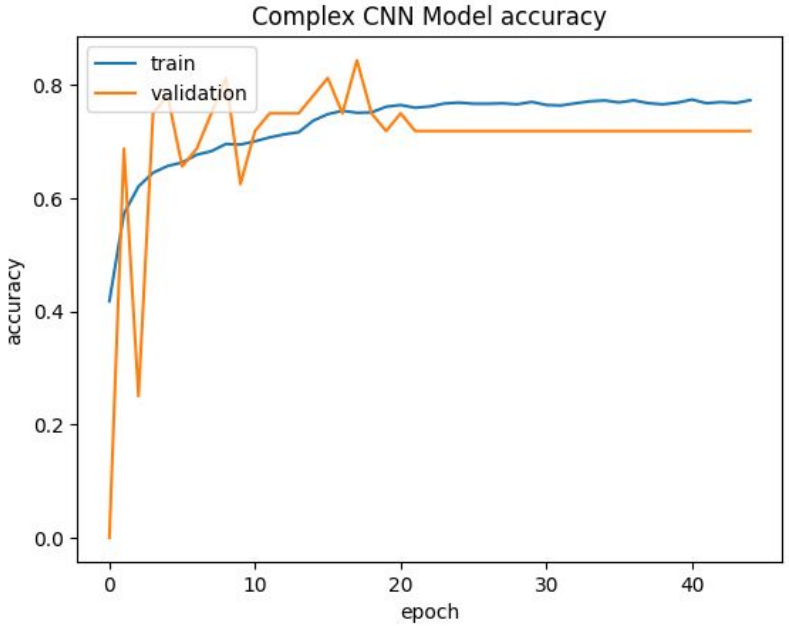
Models Comparison: Complex CNN Model

Architecture

```
# Initializing a sequential model
model_complex = Sequential()

# First Convolutional Block
model_complex.add(Conv2D(64, kernel_size=2, padding="same", input_shape=(48, 48, 1)))
model_complex.add(BatchNormalization())
model_complex.add(LeakyReLU(alpha=0.1))
model_complex.add(MaxPooling2D(pool_size=2))

# Second Convolutional Block
model_complex.add(Conv2D(128, kernel_size=2, padding="same", input_shape=(48, 48, 1)))
model_complex.add(BatchNormalization())
model_complex.add(LeakyReLU(alpha=0.1))
model_complex.add(MaxPooling2D(pool_size=2))

# Third Convolutional Block
model_complex.add(Conv2D(256, kernel_size=2, padding="same"))
model_complex.add(BatchNormalization())
model_complex.add(LeakyReLU(alpha=0.1))
model_complex.add(MaxPooling2D(pool_size=2))

# Fourth Convolutional Block
model_complex.add(Conv2D(512, kernel_size=2, padding="same"))
model_complex.add(BatchNormalization())
model_complex.add(LeakyReLU(alpha=0.1))
model_complex.add(MaxPooling2D(pool_size=2))

# Fifth Convolutional Block
model_complex.add(Conv2D(128, kernel_size=2, padding="same"))
model_complex.add(BatchNormalization())
model_complex.add(LeakyReLU(alpha=0.1))
model_complex.add(MaxPooling2D(pool_size=2))

# Flatten the output of the conv layers to feed into the dense layers
model_complex.add(Flatten())

model_complex.add(Dense(512, activation="relu"))
model_complex.add(Dense(128, activation="relu"))
model_complex.add(Dense(64))
model_complex.add(BatchNormalization())
model_complex.add(ReLU())  # Using ReLU after batch normalization
model_complex.add(Dense(4, activation="softmax"))
```

Models Comparison: EfficientNet Model

Results



	precision	recall	f1-score	support
happy	0.97	0.88	0.92	32
neutral	0.65	0.75	0.70	32
sad	0.77	0.72	0.74	32
surprise	0.91	0.91	0.91	32
accuracy			0.81	128
macro avg	0.82	0.81	0.82	128
weighted avg	0.82	0.81	0.82	128