Image Categorization Methodology using the IMFDB Dataset

Smita Darmora

University of Illinois Chicago

Intro to Machine Learning, Fall 2023

Introduction:

Primary Categories	Sub-classes
Expressions	Anger, Happiness, Sadness
Illumination	Bad, Medium, High
Pose	Frontal, Left, Right, Up, Down
Occlusions	Glasses, Beard, Ornaments
Age	Child, Young, Middle, Old
Makeup	Partial makeup, Over-makeup
Gender	Male, Female

- The project is leveraging the power of machine learning and computer vision with a focus on convolutional neural networks (CNNs) to address the complex challenge of accurately categorizing images.
- Primary focus is on the Indian Movie Face Database (IMFDB) , which contains 34,512 images exhibiting the faces of 100 Indian actors in over 100 movies.
- Aims to categorize these images into seven primary categories, each of which further branches into multiple sub-classes.
- It utilizes multiclass classification models to classify images into multiple categories using CNN.
- Additionally, pre-trained deep learning models were used, and transfer learning was applied to the features extracted from the CNN layers.
- Optimizing the number of models required for each of its seven primary categories.
- Libraries: Pandas, Matplotlib, Keras, and TensorFlow.

Data Preparation:

- Data sources and format
- Data Cleaning with Pandas
- Data Categorization
- Build an input pipeline

```python
import os
os.mkdir("train/GENDER")
os.mkdir("train/GENDER/MALE")
os.mkdir("train/GENDER/FEMALE")

os.mkdir("test/GENDER")
os.mkdir("test/GENDER/MALE")
os.mkdir("test/GENDER/FEMALE")
```

```python
for i in range(X_train.shape[0]):
  location = X_train["imagepath"][i]
  gender = X_train["Gender"][i]
  name = Path(location).name
  if gender == "MALE":
    dest = os.path.join("/content/train/GENDER/MALE",name)
  else:
    dest = os.path.join("/content/train/GENDER/FEMALE",name)
  shutil.copy(location,dest)
```

```python
# load the Train images
train_ds = tf.keras.utils.image_dataset_from_directory(
  trainPath,
  image_size=IMG_SIZE,
  seed=123,
  batch_size=batch_size,
  validation_split=0.3,
  subset="training")
```

Build the Model:

- As proposed, seven multi-class classification models using Convolutional Neural Networks (CNNs) were build.

- These models are designed to classify inputs into one of several classes, playing an instrumental role in predicting the various subclasses associated with each primary category.

- For each model, three different strategies were used:
  - First, to built a base CNN model and then
  - To improve the accuracy, two different approaches were applied
    - model architecture with batch normalization and spatial dropout,
    - and transfer learning using **ResNet50** and VGG.

- Considering there are seven primary categories to predict, our approach focuses on exploring strategies to minimize the number of necessary models.

Base CNN Architecture:

```
# CNN Model Structure
batch_size = 32
IMG_SIZE = (224, 224)
num_classes_age = len(df['Age '].unique())
print(num_classes_age)

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(num_classes_age) )

#Compile the model
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              optimizer='adam',
              metrics=['accuracy'])

# Train the model
history = model.fit(
    train_age_ds,
    epochs=20,
    validation_data=val_age_ds)
```

CNN Model Overview:

- Image Input Size: images with dimensions of 224 x 224 pixels.
- Layers Configuration: Three convolutional layers followed by max pooling layers.

Filters Detail:

- First two layers: 32 filters each, size 3 x 3.
- Third layer: 64 filters to increase complexity.

Dense Layers and Activation:

- Flattening: Post-convolutional layers output flattened for dense layer input.
- Dense Layer Configuration:
  - One dense layer with 64 neurons.
  - Activation Function: ReLU

Output Layer and Compilation

- Final Layer: Designed with four neurons corresponding to the dataset's four unique age categories.

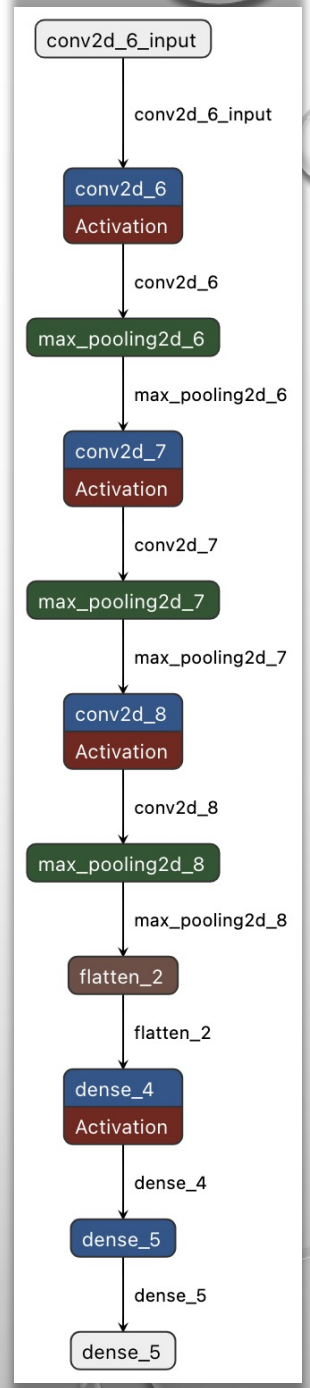
Compilation/configuration:

- Optimizer: Adam.
- Loss Function: Sparse Categorical Crossentropy

Training and Validation:

- Training Parameters: Epochs: 20 and Batch Size: 32.
- Validation: Performance assessed on a separate validation dataset.

https://netron.app/



CNN Architecture with Batch Normalization and Spatial Dropout:

```
# Model Architecture with Batch Normalization and Spatial Dropout
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(SpatialDropout2D(0.5))  # Spatial Dropout
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.001)))
model.add(BatchNormalization())
model.add(Dense(num_classes_age))

# Using SGD Optimizer with a starting learning rate
optimizer = SGD(learning_rate=0.01, momentum=0.9)

# Compile the model
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              optimizer=optimizer,
              metrics=['accuracy'])

# Callbacks: Early Stopping and Reduce Learning Rate on Plateau
early_stopping = EarlyStopping(monitor='val_loss', patience=5)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.001)

# Train the model
history = model.fit(
    train_age_ds,
    epochs=20,
    validation_data=val_age_ds,
    callbacks=[early_stopping, reduce_lr]  # Include both callbacks
```

CNN Architecture with Batch Normalization and Spatial Dropout:

- **Initial Layer:** Begins with a 2D convolutional layer, 32 filters of size 3x3, ReLU activation, tailored for 224x224 pixel images with 3 color channels.

Normalization and Pooling:

- **Batch Normalization:** Applied after the initial convolutional layer to stabilize and accelerate training.
- **Max Pooling:** 2x2 pool size used after the first two convolutional layers to reduce spatial dimensions.

Overfitting Prevention:

- **Spatial Dropout:** Implemented after the second pooling layer with a 50% dropout rate to reduce overfitting by omitting feature detectors randomly.

Deepening the Network:

- **Further Convolution:** Additional convolutional layer with 64 filters followed by batch normalization and max pooling.
- **Flattening:** Flattens output for dense layer processing.

Dense Layer and Regularization:

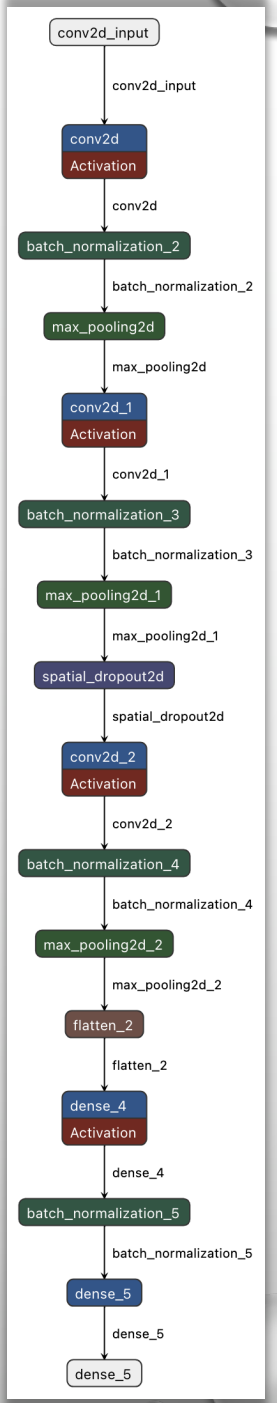
- **Dense Layer:** 64 neurons with ReLU activation and L2 regularization (lambda = 0.001).
- **Subsequent Normalization:** Another batch normalization before the final output layer.

Final Layer and Optimization:

- **Output Layer:** Final dense layer for unique age category classification.
- **Optimizer:** Stochastic Gradient Descent with an initial learning rate of 0.01 and momentum of 0.9.

Compilation and Callbacks:

- **Loss Function:** Sparse Categorical Crossentropy.
- **Performance Metric:** Accuracy.
- **Enhancements:** Early stopping and reduced learning rate on plateau for improved training.



CNN Architecture leveraging Transfer Learning (ResNet50):

```python
# Load the ResNet50 model, pre-trained on ImageNet data, without the top layer
batch_size = 32
IMG_SIZE = (224, 224)
num_classes_age = len(df['Age '].unique())
print(num_classes_age)

base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freeze the layers of the base model
for layer in base_model.layers:
    layer.trainable = False

# Create a new model on top
model = Sequential([
    base_model,
    Flatten(),
    Dense(64, activation='relu', kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    Dropout(0.5),
    Dense(num_classes_age, activation='softmax')
])

# Using SGD Optimizer with a starting learning rate
optimizer = SGD(learning_rate=0.01, momentum=0.9)

# Compile the model
model.compile(loss='sparse_categorical_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])

# Callbacks: Early Stopping and Reduce Learning Rate on Plateau
early_stopping = EarlyStopping(monitor='val_loss', patience=5)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=4, min_lr=0.001)

# Train the model
history = model.fit(
    train_age_ds,
    epochs=20,
    validation_data=val_age_ds,
    callbacks=[early_stopping, reduce_lr]
```

Leveraging ResNet50 for Classification

- **Core Architecture:** Utilizes ResNet50, pre-trained on ImageNet.
- **Image Input Size:** Configured for 224x224 pixel images.
- **Layer Adaptation:** Top layer removed to fit specific classification needs.

Freezing Base Model Layers

- **Preserving Pre-trained Features:** Base ResNet50 layers are frozen.
- **Trainable Layers:** Only newly added layers are set to be trainable to fine-tune the model for the new dataset.

Extended Architecture

- **Flatten Layer:** Converts 2D feature maps into a 1D vector.
- **Dense Layer:** Adds complexity with 64 neurons, ReLU activation, and L2 regularization (lambda = 0.001).

Enhancing Generalization

- **Batch Normalization:** Applied post dense layer to stabilize learning.
- **Dropout Layer:** Set at 50% to prevent overfitting and enhance model generalization.

Output Layer Configuration

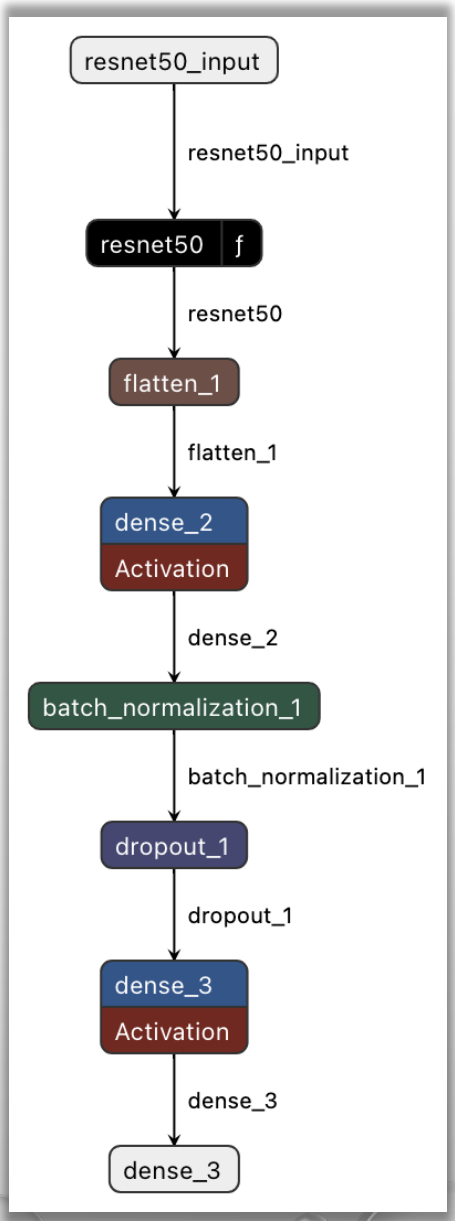
- **Softmax Activation:** Employed in the final dense layer for multi-class classification.
- **Neurons Count:** Corresponds to the number of unique categories in the dataset.

Optimization Techniques

- **Optimizer:** SGD with an initial learning rate of 0.01 and momentum of 0.9.
- **Loss Function:** Sparse Categorical Crossentropy, targeting accuracy.

Training Refinements

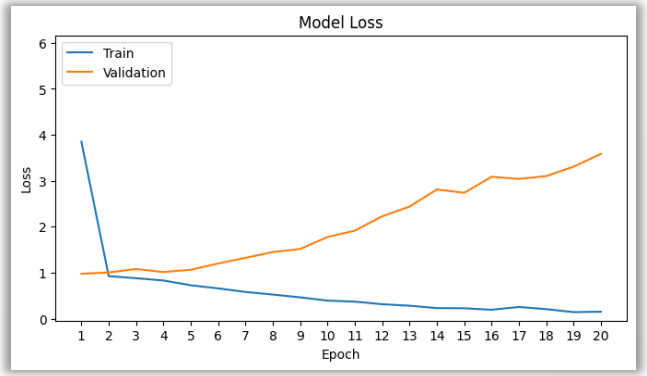
- **Early Stopping:** Monitors validation loss to optimize the training process and prevent overfitting.
- **Training and Validation:** Conducted on a designated dataset with validation data for performance assessment.

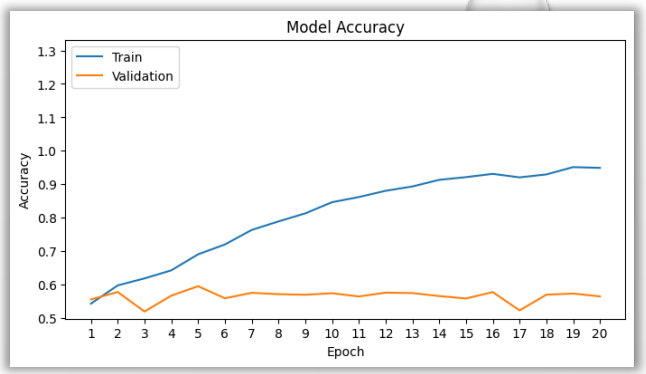


Visualize training results:

Base CNN Model Performance:

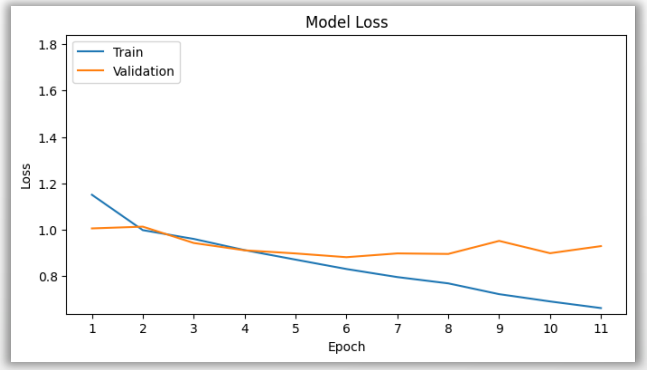
- **Training Accuracy:** High at approximately 94%.
- **Validation Accuracy:** Comparatively lower at around 56%, indicating potential overfitting.

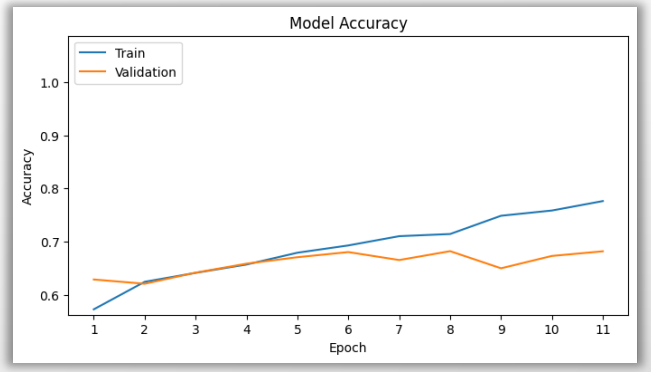




Improved Model with Batch Normalization and Spatial Dropout:

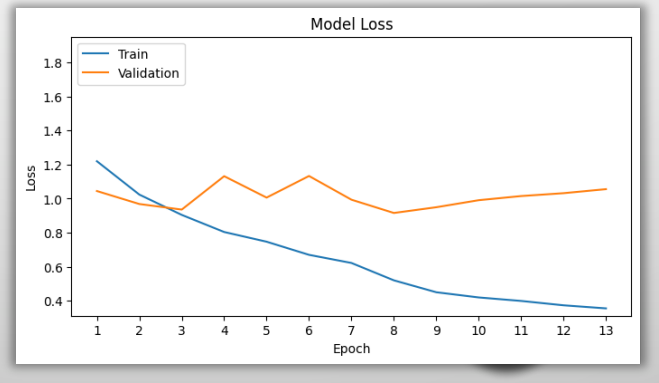
- **Adjusted Training Accuracy:** Decreased to 80%, suggesting reduced overfitting.
- **Enhanced Validation Accuracy:** Improved to approximately 71%, indicating better generalization.

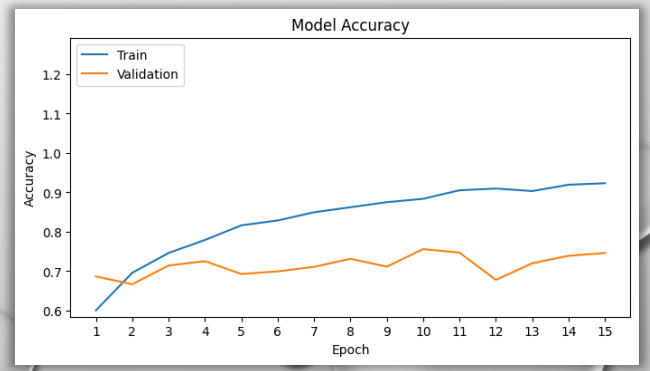




Transfer Learning with ResNet50:

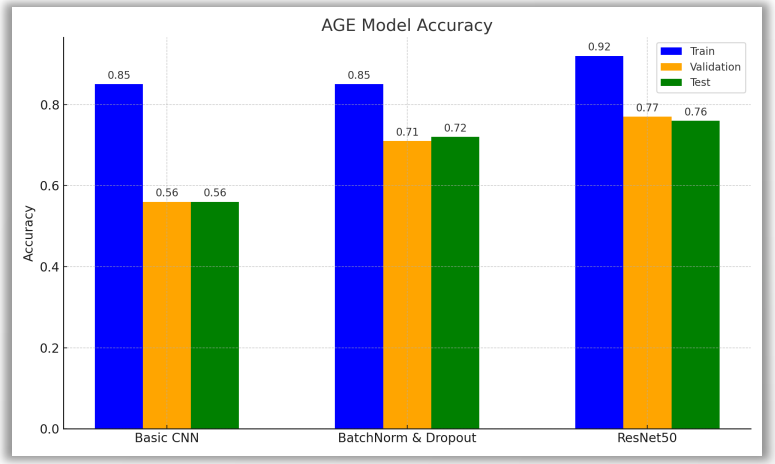
- **Training Accuracy Post-Transfer Learning:** Maintained high at around 95%.
- **Enhanced Validation Accuracy:** Improved to approximately 77%.





Predict on new data:

- Assessing the model's real-world efficacy and generalization.
- **Focus on 'Age' Model:** Specific examination of training, validation, and testing accuracies.



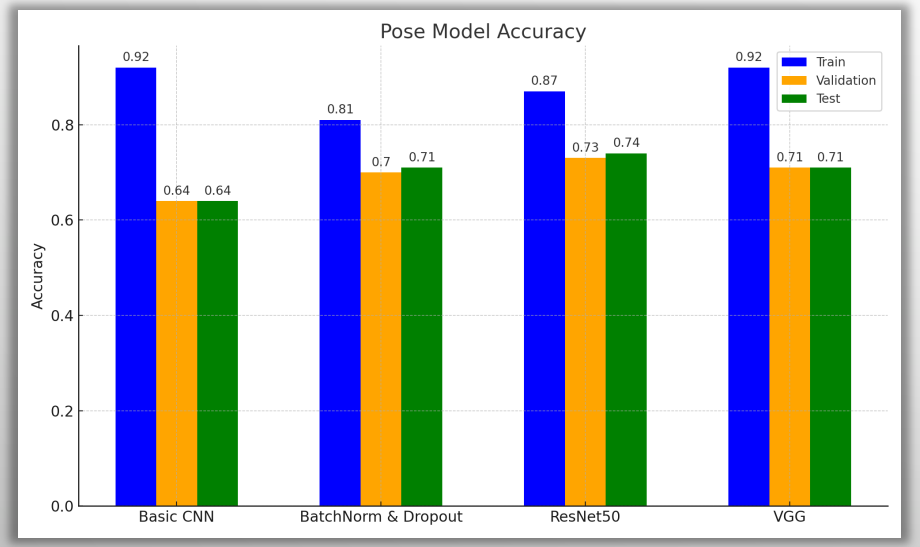
Comparative Accuracy Results

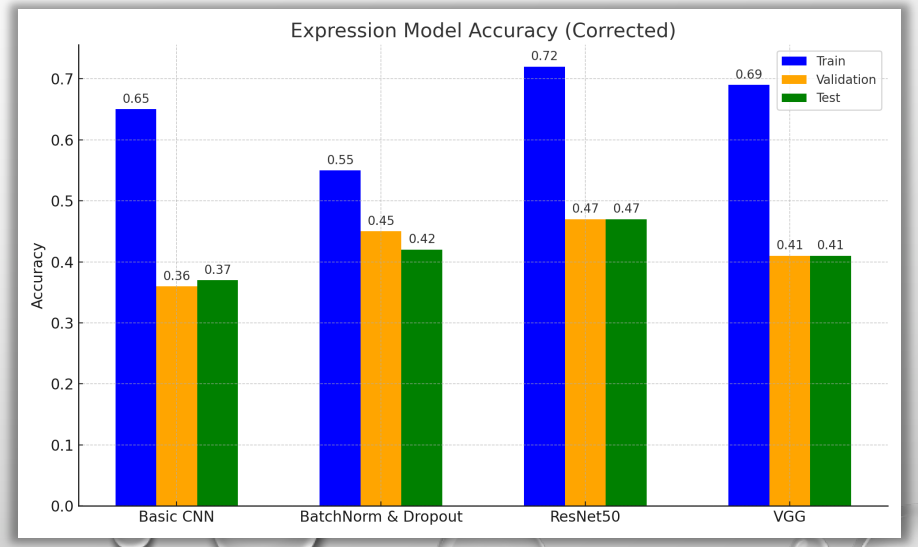
- **Base CNN Model:** Testing accuracy recorded at approximately 57%.
- **Improved CNN Model with Normalization and Dropout:** Testing accuracy improved to about 69%.
- **Transfer Learning Impact (ResNet50):** Significant validation accuracy increase to around 76%.

	Basic CNN Model			Model Architecture with BN and SD			ResNet50 Model		
	Train	Validation	Test	Train	Validation	Test	Train	Validation	Test
GENDER	0.92	0.78	0.77	0.69	0.70	0.78	0.95	0.90	0.90
MAKEUP	0.97	0.96	0.97	0.98	0.97	0.97	0.98	0.96	0.97
EXPRESSION	0.65	0.36	0.37	0.55	0.45	0.42	0.72	0.47	0.47
POSE	0.92	0.64	0.64	0.81	0.70	0.71	0.85	0.73	0.74
OCCLUSION	0.82	0.66	0.66	0.83	0.74	0.74	0.94	0.79	0.80
AGE	0.85	0.56	0.56	0.85	0.71	0.72	0.92	0.77	0.76
ILLUMINATION	0.82	0.62	0.63	0.74	0.69	0.71	0.85	0.71	0.72
AGE_GENDER	0.90	0.54	0.54	0.87	0.67	0.68	0.89	0.75	0.75
MAKEUP_ILLUMINATION	0.81	0.61	0.62	0.79	0.68	0.69	0.86	0.69	0.70

VGG vs ResNet50:

- For transfer learning, initially experimented with two well-recognized CNN architectures: VGG and ResNet50.
- Architectures were applied to two distinct models: 'Pose' and 'Expression'.
- In the comparative analysis, ResNet50 demonstrated superior performance compared to VGG, in accuracy
- Therefore, decided to implement ResNet50 across all models.





Optimizing Model Count:

Model Optimization Strategy

- Initial assumption: Seven models for seven primary categories and reduce model count by combining categories.
- Ex: 'Age' and 'Gender' merged into 'Gender Age'; 'Makeup' and 'Illumination' combined into 'Makeup_Illumination'.

New Combined Categories

- 'Age_Gender' Model: Combines four age groups (Child, Young, Middle, Old) with two gender categories (Male, Female), resulting in seven distinct categories.
  - Age_Gender' categories: Male Child, Male Young, Male Middle, Male Old, Female Young, Female Middle, Female Old (Note: No data for 'Female Child').
- 'Makeup_Illumination' Model: Merges two makeup categories (Partial, Over) with three illumination levels (Bad, Medium, High), leading to six combinations.
  - 'Makeup_Illumination' categories: Partial Bad, Partial Medium, Partial High, Over Bad, Over Medium, Over High.

Implementation of New Models

- Base CNN Model developed with enhancements like Batch Normalization and Spatial Dropout.
- Transfer learning with ResNet50 employed for improved accuracy.
- Only two models created due to lack of other meaningful combinations.

Conclusion:

Objective of the Study

- Utilize the IMFDB for advanced image categorization.
- Employ convolutional neural networks to classify images into seven categories.
- Incorporate multiclass classification and transfer learning to boost efficiency and accuracy.

key learnings and outcomes of the study

Importance of Data Preparation

- Critical role of data understanding, preparation, and cleaning.
- Data preprocessing is essential for model effectiveness.

Lessons on CNN Architecture

- The tendency for overtraining with basic CNN architecture as evidenced by lower testing accuracy.
- Importance of overfitting to improve model performance.

Enhancing Model Accuracy

- Batch normalization and spatial dropout as effective techniques for accuracy improvement.
- Transfer learning's role in further enhancing model accuracy.

Performance Comparison

- ResNet50 outperforms VGG, consistent with existing literature.
- Trade-offs between combining models for optimized model count versus potential accuracy reduction.