Transfer Learning Report

# 1. Transfer Learning Feature Extraction (Without Data Augmentation)

Introduction:  
Transfer learning is a machine learning technique where a model developed for a specific task is reused as the starting point for a model on a second task. This notebook demonstrates the process of using transfer learning for feature extraction without applying data augmentation techniques. The goal is to leverage pre-trained models to extract features from a new dataset and use these features to train a classifier.

## Steps:

1. Loading Pre-trained Model:  
```python  
from tensorflow.keras.applications import VGG16  
base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))  
```

2. Feature Extraction:  
```python  
features = base\_model.predict(new\_data)  
```

3. Training Classifier:  
```python  
from sklearn.linear\_model import LogisticRegression  
classifier = LogisticRegression()  
classifier.fit(features, labels)  
```

4. Evaluation:  
```python  
from sklearn.metrics import accuracy\_score  
predictions = classifier.predict(test\_features)  
print('Accuracy:', accuracy\_score(test\_labels, predictions))  
```

## Observations:

- Feature extraction significantly reduces training time since the base model is not retrained.  
- The model's performance is dependent on the quality of the pre-trained features.

# 2. Transfer Learning Feature Extraction (With Data Augmentation)

Introduction:  
Data augmentation is a technique to artificially increase the size of the training dataset by creating modified versions of the existing data. This notebook extends the feature extraction process by incorporating data augmentation techniques to enhance the training data.

## Steps:

1. Data Augmentation:  
```python  
from tensorflow.keras.preprocessing.image import ImageDataGenerator  
datagen = ImageDataGenerator(rotation\_range=40, width\_shift\_range=0.2, height\_shift\_range=0.2, shear\_range=0.2, zoom\_range=0.2, horizontal\_flip=True, fill\_mode='nearest')  
```

2. Loading Pre-trained Model:  
(Similar to the previous notebook.)

3. Feature Extraction with Augmentation:  
```python  
augmented\_data = datagen.flow\_from\_directory('data/train', target\_size=(224, 224), batch\_size=32, class\_mode='binary')  
features = base\_model.predict(augmented\_data)  
```

4. Training Classifier:  
Train the classifier on augmented features.

5. Evaluation:  
Evaluate the augmented model's performance.

## Observations:

- Data augmentation improves model generalization, leading to better performance on test data.  
- The augmented model shows higher accuracy compared to the non-augmented version.

# 3. Transfer Learning Fine-tuning

Introduction:  
Fine-tuning involves unfreezing some of the layers of the pre-trained model and retraining them on the new dataset, allowing the model to adapt to the specific features of the new data.

## Steps:

1. Loading Pre-trained Model:  
```python  
from tensorflow.keras.applications import VGG16  
base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))  
```

2. Freezing Initial Layers:  
```python  
for layer in base\_model.layers[:15]:  
 layer.trainable = False  
```

3. Adding New Layers:  
```python  
from tensorflow.keras.models import Model  
from tensorflow.keras.layers import Dense, Flatten  
x = base\_model.output  
x = Flatten()(x)  
x = Dense(1024, activation='relu')(x)  
predictions = Dense(1, activation='sigmoid')(x)  
model = Model(inputs=base\_model.input, outputs=predictions)  
```

4. Compiling and Training:  
```python  
model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])  
model.fit(train\_data, epochs=5, validation\_data=val\_data)  
```

5. Evaluation:  
```python  
loss, accuracy = model.evaluate(test\_data)  
print('Test accuracy:', accuracy)  
```

## Observations:

- Fine-tuning allows the model to adapt specifically to the new dataset, resulting in improved performance.  
- Requires more computational resources and time compared to feature extraction alone.

# Comparative Analysis

## Performance Comparison:

-\*Feature Extraction (Without Augmentation):Quick and efficient, suitable for datasets with similar features to the pre-trained model's training data.  
- Feature Extraction (With Augmentation): Better performance due to increased data variability, making the model more robust.  
- Fine-tuning: Best performance for complex tasks and datasets that differ significantly from the pre-trained model's training data.

## Training Time:

- Without Augmentation: Fastest, minimal computational overhead.  
- With Augmentation:Increased training time due to larger augmented dataset.  
- Fine-tuning: Longest training time, as more layers are retrained.

#### 1. Transfer Learning with Fine-tuning

**Objective and Dataset:**

* **Objective:** Fine-tune a pre-trained VGG16 model on the Dogs vs. Cats dataset.
* **Dataset:** Dogs vs. Cats dataset from Kaggle.

**Model Architecture and Pre-trained Weights:**

* **Model:** VGG16 model pre-trained on ImageNet.
* **Pre-trained Weights:** Loaded with weights from ImageNet.

**Methodology:**

* **Data Preparation:**
  + Downloaded the dataset from Kaggle.
  + Extracted the dataset into a suitable directory.
* **Model Configuration:**
  + Initialized the VGG16 model without the top classifier layer (include\_top=False).
  + Set layers up to 'block5\_conv1' as non-trainable to retain early learned features.
  + Added new dense layers for binary classification.
* **Data Augmentation and Preprocessing:**
  + Created training and validation datasets using image\_dataset\_from\_directory.
  + Normalized the images by scaling pixel values.
* **Training:**
  + Compiled the model with RMSprop optimizer and binary cross-entropy loss.
  + Trained for 10 epochs.

**Results and Performance Metrics:**

* The notebook ends with a plot of training and validation accuracy and loss.

**Observations and Conclusions:**

* **Performance Insights:** The training and validation metrics provide insight into model performance and overfitting.

#### 2. Transfer Learning with Feature Extraction (Data Augmentation)

**Objective and Dataset:**

* **Objective:** Use a pre-trained VGG16 model for feature extraction with data augmentation on the Dogs vs. Cats dataset.
* **Dataset:** Dogs vs. Cats dataset from Kaggle.

**Model Architecture and Pre-trained Weights:**

* **Model:** VGG16 model pre-trained on ImageNet.
* **Pre-trained Weights:** Used for feature extraction.

**Methodology:**

* **Data Preparation:**
  + Similar steps for downloading and extracting the dataset.
* **Model Configuration:**
  + Initialized VGG16 without the top classifier layer.
  + Added custom dense layers for binary classification.
  + Set the entire pre-trained model as non-trainable.
* **Data Augmentation and Preprocessing:**
  + Training and validation datasets created without explicit data augmentation.
  + Normalized the images.
* **Training:**
  + Compiled the model with Adam optimizer and binary cross-entropy loss.
  + Trained for 10 epochs.

**Results and Performance Metrics:**

* Plots of training and validation accuracy and loss were included to show model performance.

**Observations and Conclusions:**

* **Performance Insights:** Provides a comparison point with and without data augmentation in terms of model performance.

#### 3. Transfer Learning with Feature Extraction (Without Data Augmentation)

**Objective and Dataset:**

* **Objective:** Use a pre-trained VGG16 model for feature extraction without data augmentation on the Dogs vs. Cats dataset.
* **Dataset:** Dogs vs. Cats dataset from Kaggle.

**Model Architecture and Pre-trained Weights:**

* **Model:** VGG16 model pre-trained on ImageNet.
* **Pre-trained Weights:** Used for feature extraction.

**Methodology:**

* **Data Preparation:**
  + Similar steps for downloading and extracting the dataset.
* **Model Configuration:**
  + Initialized VGG16 without the top classifier layer.
  + Added custom dense layers for binary classification.
  + Set the entire pre-trained model as non-trainable.
* **Data Preprocessing:**
  + Created training and validation datasets.
  + Normalized the images by scaling pixel values.
* **Training:**
  + Compiled the model with Adam optimizer and binary cross-entropy loss.
  + Trained for 10 epochs.

**Results and Performance Metrics:**

* Plots of training and validation accuracy and loss were included to show model performance.

**Observations and Conclusions:**

* **Performance Insights:** Allows for a direct comparison with the augmented dataset results to evaluate the impact of data augmentation.

Lastly we were also asked to insert frog images for which we didn’t get most appropriate result as the dataset we uploaded for training had few images of frog compared to dogs and cats.

# Conclusion

These notebooks provide a comprehensive guide on applying transfer learning techniques for different scenarios:  
- Use feature extraction for quick, efficient model training.  
- Apply data augmentation for improved model robustness.  
- Opt for fine-tuning when high performance on a specific dataset is required.