Machine Learning Final Project

Knee Osteoarthritis Prediction Using Deep Learning

By Ali Irtaza, Sri Sankeerth and Anantha

The George Washington University

Computer Science

CS-6364

Table of Content

Introduction

This project aims to develop an automated system for classifying the severity of knee osteoarthritis (OA) using X-ray images. We employ advanced deep learning models (VGG19 and Xception) with transfer learning and customized weights to achieve a 5-class classification scheme. To prepare the images for optimal model performance, we implement custom image processing techniques.

Data

Dataset

The dataset was collected from two unique sources. The training data comprises 9786 datapoints distributed across three classes. This training data is split into five classes with noticeable class imbalance, as illustrated in Image 1. For validation and testing, 7828 datapoints are used, with 65% allocated for validation and 35% for testing. Image 2 demonstrates the class imbalance present in this validation and test data split.

Image 1: Train Data Distribution

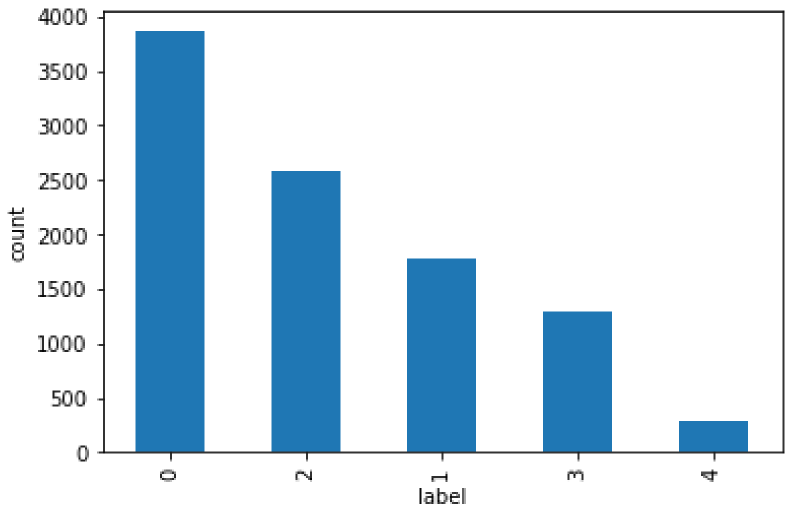
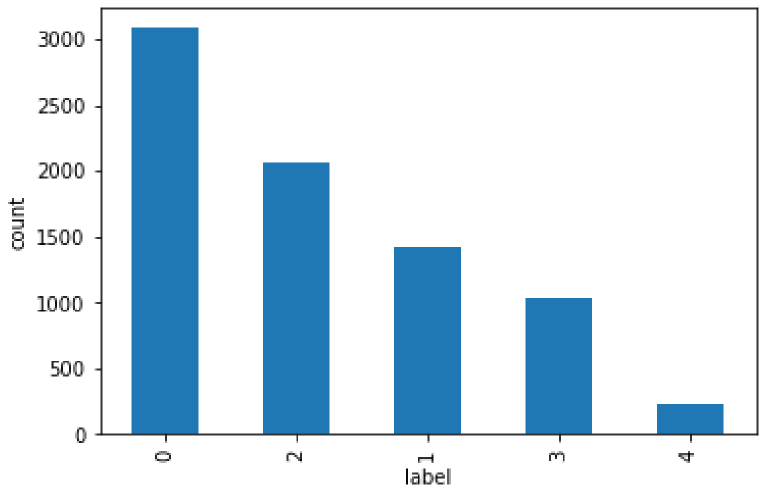


Image 2: Validation and Test Distribution



Data Pre-Processing

The images were preprocessed in several steps to prepare them for analysis. First, they were resized to a consistent dimension and converted to grayscale for simplified processing. Pixel values were then normalized to standardize brightness and contrast across the dataset. To reduce noise, Gaussian blurring was applied, followed by histogram equalization to enhance the visibility of important features. This preprocessing methodology likely aims to optimize the images for use in a machine learning model.

Before Pre-Processing

Image 3: Class 0

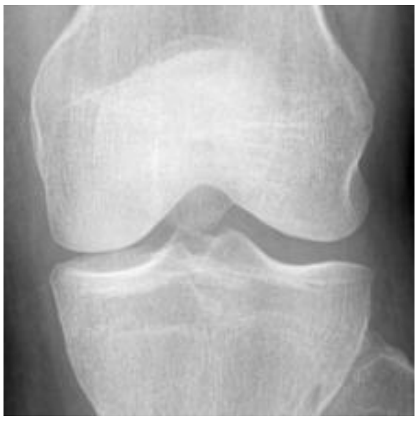


Image 4: Class 1

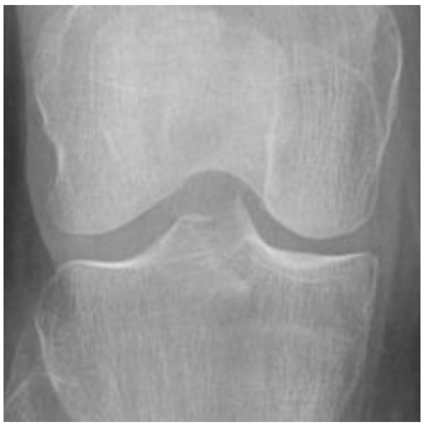


Image 5: Class 2

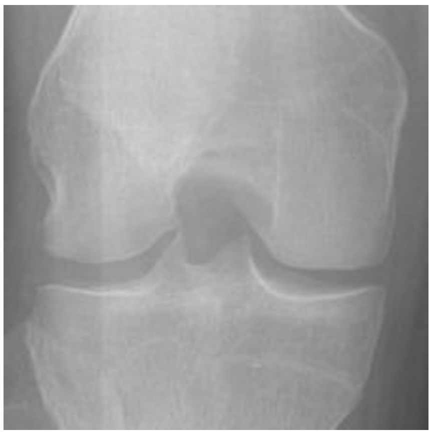


Image 6: Class 3

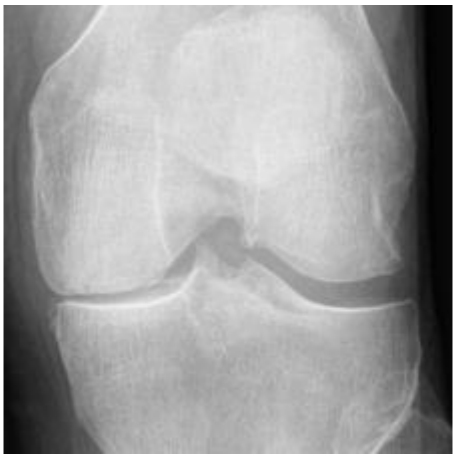
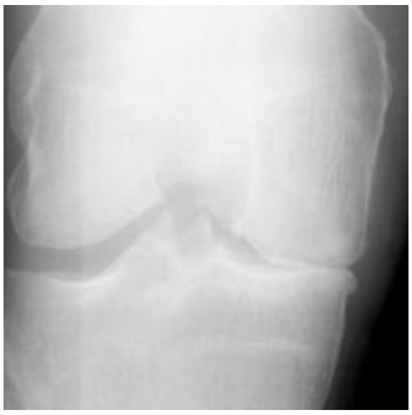


Image 7: Class 4



After Pre-Processing

Image 8: Class 0

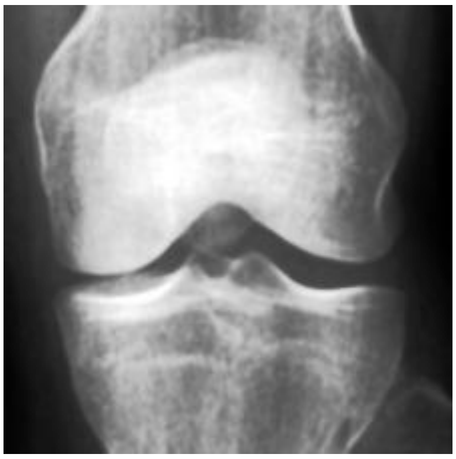


Image 9: Class 1

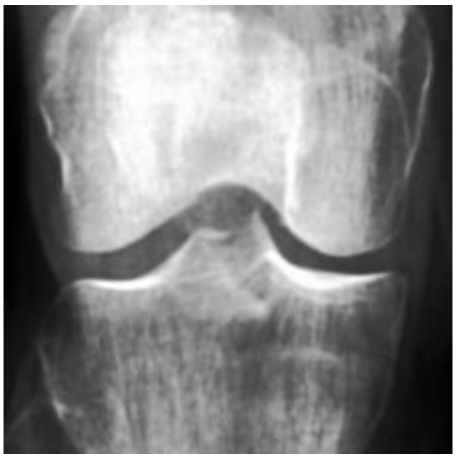


Image 10: Class 2

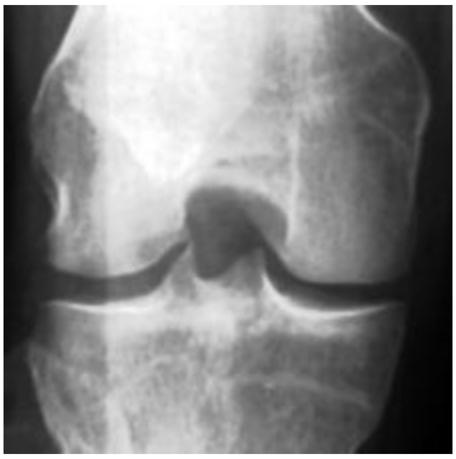


Image 11: Class 3

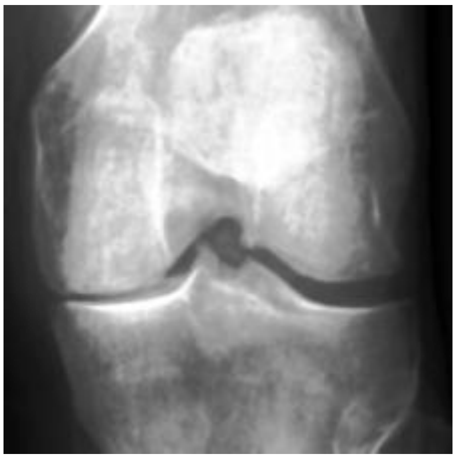
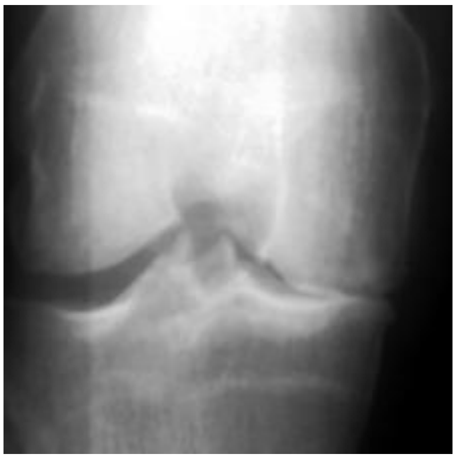


Image 12: Class 4



Models

Baseline Model

Model

Our group decided to build our project foundation on the powerful VGG19 convolutional neural network (CNN). Known for its excellence in image classification, VGG19 provided a strong starting point. We used a basic transfer learning approach, following standard practice by using the model's default hyperparameters. The substantial parameter count – 20,156,997 in total, with 133,613 trainable – highlights the complexity of this architecture.

To streamline training, we strategically employed VGG19's pre-trained ImageNet weights. These weights embody the knowledge gained from extensive training on a massive image dataset. By freezing them, we could focus our training efforts solely on the new classification layers we added.

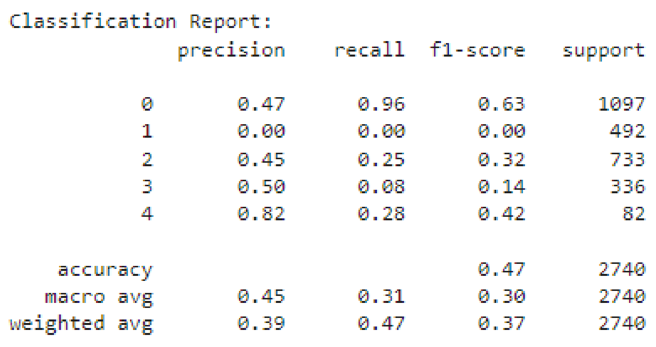
Our custom classification head maintained a relatively straightforward design. First, a GlobalAveragePooling2D layer condenses the output of VGG19's feature extractor, reducing dimensionality and helping to mitigate overfitting. Next, we added two dense (fully connected) layers with ReLU activations. These layers work together to learn how to combine the extracted features in a way that aligns with the five classes present within our project's dataset. The model culminates in a Dense layer with five output neurons (matching the number of classes) and utilizes the softmax activation function to generate our final class probabilities.

We selected the 'adam' optimizer for training due to its reliability and widespread use in neural network optimization. For our multi-class classification problem with integer-based class labels, the 'sparse_categorical_crossentropy' loss function was the clear choice. To monitor our progress, we tracked the 'accuracy' metric, which measures the percentage of correctly classified examples.

Metrics

This classification report reveals performance difficulties in classifying classes 1, 2, and 3, as highlighted by the F1-scores in Table 1. The significant class imbalance underscores the importance of the F1-score, which offers a more robust evaluation metric than accuracy alone. Epoch 8 (Image 3) likely represents the optimal training point; further iterations could lead to overfitting and reduced generalization performance. This pattern is reinforced by the confusion matrix (Image 4), which emphasizes misclassifications within classes 1, 2, and 3. To enhance results, consider exploring alternative feature engineering techniques, model architectures, or hyperparameter optimization.

Table 1: Classification Report for the Baseline Model



```
Classification Report:
              precision    recall  f1-score   support

           0       0.47      0.96      0.63      1097
           1       0.00      0.00      0.00       492
           2       0.45      0.25      0.32       733
           3       0.50      0.08      0.14       336
           4       0.82      0.28      0.42        82

    accuracy                           0.47      2740
   macro avg       0.45      0.31      0.30      2740
weighted avg       0.39      0.47      0.37      2740
```

Image 13: Bias Variance Tradeoff for the Baseline Model

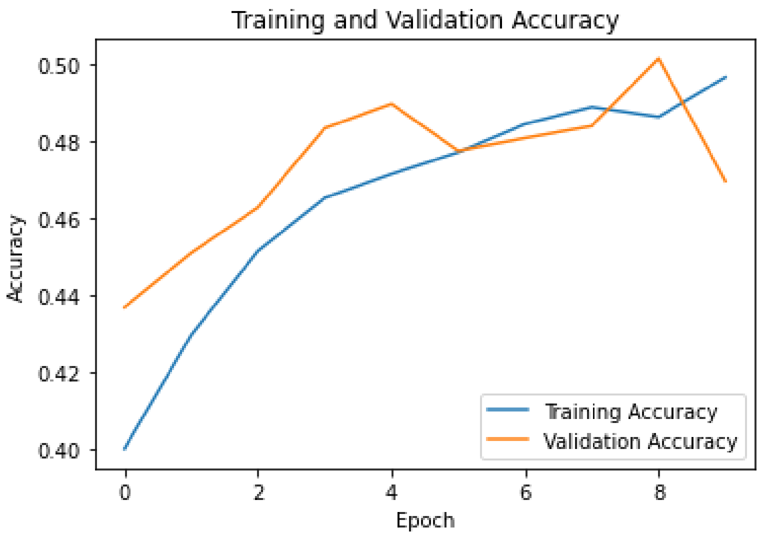
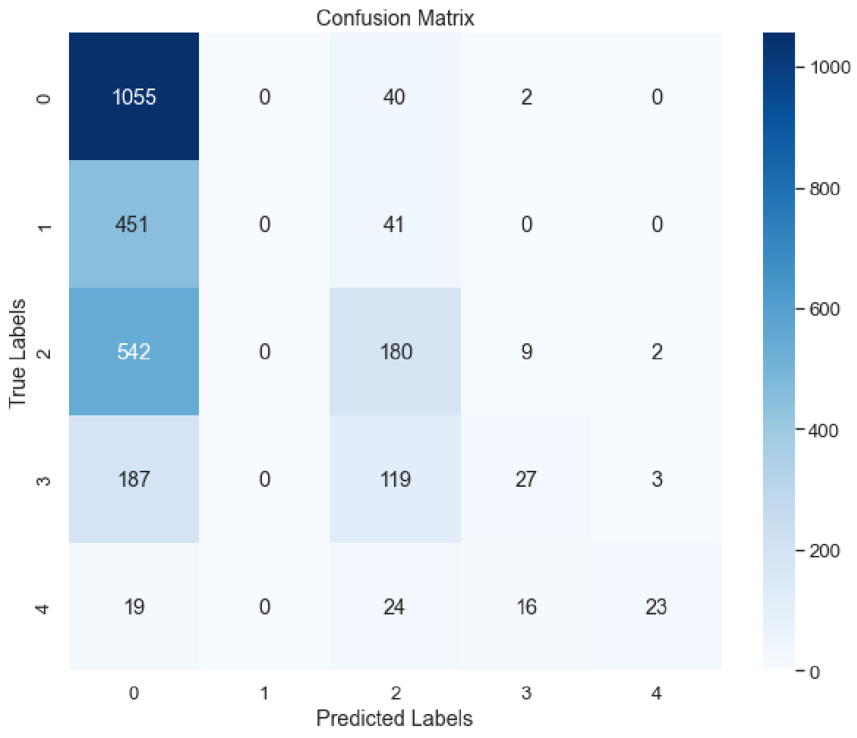


Image 14: Confusion Matrix for the Baseline Model



	0	1	2	3	4
0	1055	0	40	2	0
1	451	0	41	0	0
2	542	0	180	9	2
3	187	0	119	27	3
4	19	0	24	16	23

VGG19 Model

Callbacks and Checkpoints

Firstly, the ModelCheckpoint callback is configured to monitor the validation accuracy (val_acc) during training. It saves the model's weights to the specified checkpoint path (vgg19_best.ckpt), ensuring that only the best-performing weights are saved (save_best_only=True). This provides crucial protection against data loss and helps retain the model's progress in the event of interruptions.

Secondly, the EarlyStopping callback is employed to halt training if the validation loss (val_loss) stops improving for a specified number of epochs (patience=5). This mechanism is essential for preventing overfitting, as it terminates training when the model's performance on unseen data starts to degrade.

Lastly, the ReduceLROnPlateau callback dynamically adjusts the learning rate during training. By monitoring the validation loss (val_loss), it reduces the learning rate if no improvement is observed for a certain number of epochs (patience=5). This aids in fine-tuning the training process, allowing the optimizer to converge more efficiently towards the optimal solution and improving overall model performance.

Together, these callbacks work in synergy to optimize the VGG19 model's training process. They ensure effective convergence, mitigate the risk of overfitting, and maximize the model's performance on unseen data.

Model

To improve upon the baseline VGG19 model, we've made several enhancements. Firstly, we employ a refined transfer learning strategy by loading the pre-trained VGG19 model with ImageNet weights and freezing its layers. This allows us to harness the powerful feature extraction capabilities learned on the ImageNet dataset, while focusing training efforts on adapting the model for our specific classification task. In addition, we replaced the original VGG19 classification head with a custom one. This custom head is designed to learn highly discriminative features tailored to our dataset. It consists of four convolutional layers, each followed by batch normalization. The convolutional layers progressively distill more complex and abstract patterns, while batch normalization improves training stability and reduces the risk of overfitting. This customized architecture allows the model to effectively capture the unique characteristics of our image data, enhancing its classification capabilities.

Furthermore, we introduce custom weights calculated using the class_weight.compute_class_weight function from the sklearn.utils module. These weights address potential class imbalances within our dataset. By assigning higher weights to minority classes, we ensure that the model doesn't overlook them during the training process. This focus on balanced class representation helps the model generalize better to unseen data.

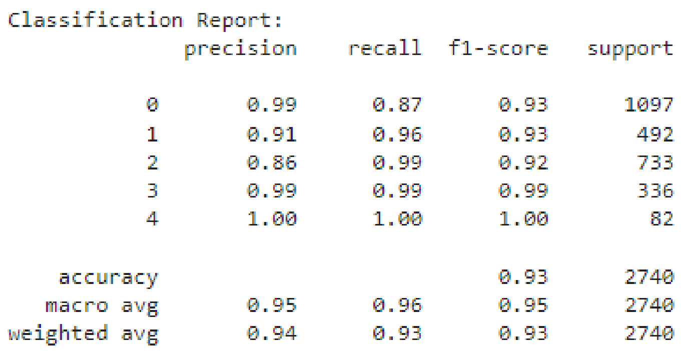
Regarding optimization, we use the Adam optimizer with a carefully chosen learning rate (0.00001) and weight decay (0.0001). The low learning rate aids in fine-tuning the pre-trained model, and weight decay encourages generalization by favoring simpler solutions. For our multi-class classification problem with integer-based class labels, the 'sparse_categorical_crossentropy' loss function was the clear choice. To monitor our progress, we tracked the 'accuracy' metric, which measures the percentage of correctly classified examples.

Metrics

In comparison to the baseline model, there seem to be substantial improvements not only in accuracy but also in the F1-scores across all classes. This classification report reveals good performance on all 5 of the classes, as highlighted by the F1-scores in Table 2. The significant class imbalance underscores the importance of the F1-score, which offers a more robust evaluation metric than accuracy alone. While the model achieves its optimal performance at

22 epochs earlier epochs may still be suitable (Image 5). Further training risks overfitting and potentially reduced generalization. The confusion matrix (Image 6) reinforces this finding, showing that the model can classify all classes relatively well.

Table 2: Classification Report for the VGG19 Model



```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.87      0.93      1097
           1       0.91      0.96      0.93       492
           2       0.86      0.99      0.92       733
           3       0.99      0.99      0.99       336
           4       1.00      1.00      1.00        82

    accuracy                           0.93      2740
   macro avg       0.95      0.96      0.95      2740
weighted avg       0.94      0.93      0.93      2740
```

Image 15: Bias Variance Tradeoff for the VGG19

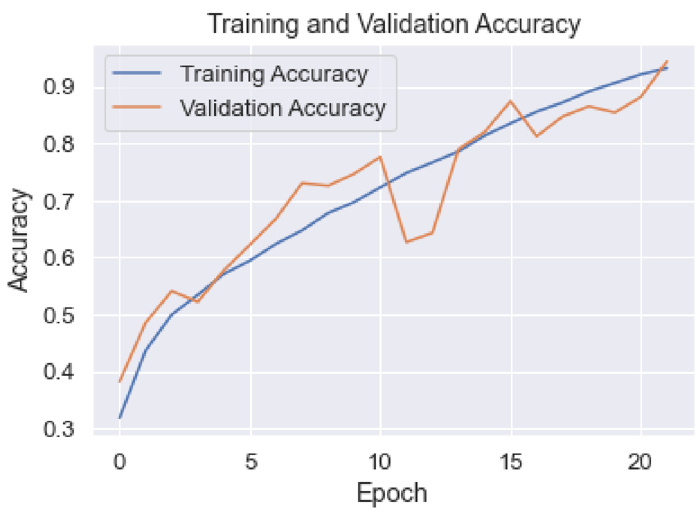
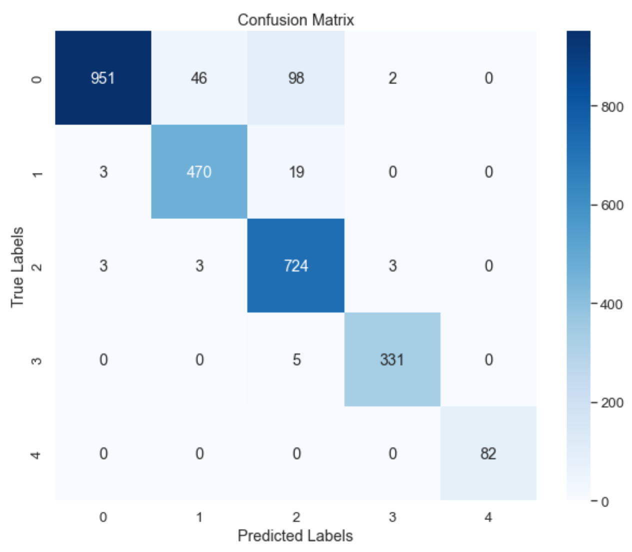


Image 16: Confusion Matrix for the VGG19 Model



Xception Model

Callbacks and Checkpoints

Firstly, the ModelCheckpoint callback is configured to monitor the validation accuracy (val_acc) during training. It saves the model's weights to the specified checkpoint path (xception_best.ckpt), ensuring that only the best-performing weights are saved (save_best_only=True). This provides crucial protection against data loss and helps retain the model's progress in the event of interruptions.

Secondly, the EarlyStopping callback is employed to halt training if the validation loss (val_loss) stops improving for a specified number of epochs (patience=5). This mechanism is essential for preventing overfitting, as it terminates training when the model's performance on unseen data starts to degrade.

Lastly, the ReduceLROnPlateau callback dynamically adjusts the learning rate during training. By monitoring the validation loss (val_loss), it reduces the learning rate if no improvement is observed for a certain number of epochs (patience=5). This aids in fine-tuning the training process, allowing the optimizer to converge more efficiently towards the optimal solution and improving overall model performance.

Together, these callbacks work in synergy to optimize the Xception model's training process. They ensure effective convergence, mitigate the risk of overfitting, and maximize the model's performance on unseen data.

Model

To improve upon the VGG19 model, we've incorporated strategic advancements into the Xception model. Firstly, it capitalizes on the power of depthwise separable convolutions. This technique significantly decreases computational cost and parameter count compared to traditional convolutions in the VGG19. This reduction in complexity helps mitigate overfitting and accelerate training. Moreover, we leverage a refined transfer learning approach by loading pre-trained Xception weights from ImageNet. This enables us to harness robust, generalized features while fine-tuning for our specific task. Additionally, we replaced the original classification head with a custom one. This custom head includes convolutional layers with batch normalization, tailored for highly discriminative feature learning from our dataset. This customization promotes effective extraction of our image data's unique patterns, boosting classification performance.

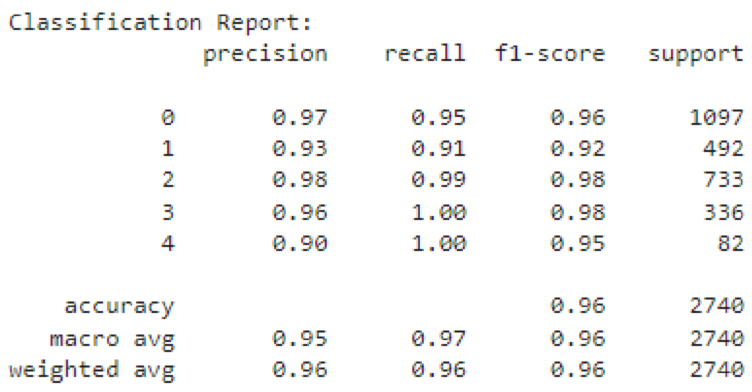
Furthermore, we introduce custom weights calculated using the class_weight.compute_class_weight function from the sklearn.utils module. These weights address potential class imbalances within our dataset. By assigning higher weights to minority classes, we ensure that the model doesn't overlook them during the training process. This focus on balanced class representation helps the model generalize better to unseen data.

Regarding optimization, we use the Adam optimizer with a carefully chosen learning rate (0.00001) and weight decay (0.0001). The low learning rate aids in fine-tuning the pre-trained model, and weight decay encourages generalization by favoring simpler solutions. For our multi-class classification problem with integer-based class labels, the 'sparse_categorical_crossentropy' loss function was the clear choice. To monitor our progress, we tracked the 'accuracy' metric, which measures the percentage of correctly classified examples.

<u>Metrics</u>

In comparison to the VGG19 model, there seem to be improvements not only in accuracy but also in the F1-scores across all classes. This classification report reveals good performance on all 5 of the classes, as highlighted by the F1-scores in Table 3. The significant class imbalance underscores the importance of the F1-score, which offers a more robust evaluation metric than accuracy alone. While the model achieves its optimal performance at 20 epochs earlier epochs may still be suitable (Image 7). Further training risks overfitting and potentially reduced generalization. The confusion matrix (Image 8) reinforces this finding, showing that the model can classify all classes relatively well.

Table 3: Classification Report for the Xception Model



```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.95      0.96      1097
           1       0.93      0.91      0.92       492
           2       0.98      0.99      0.98       733
           3       0.96      1.00      0.98       336
           4       0.90      1.00      0.95        82

    accuracy                           0.96      2740
   macro avg       0.95      0.97      0.96      2740
weighted avg       0.96      0.96      0.96      2740
```

Image 17: Bias Variance Tradeoff for the Xception Model

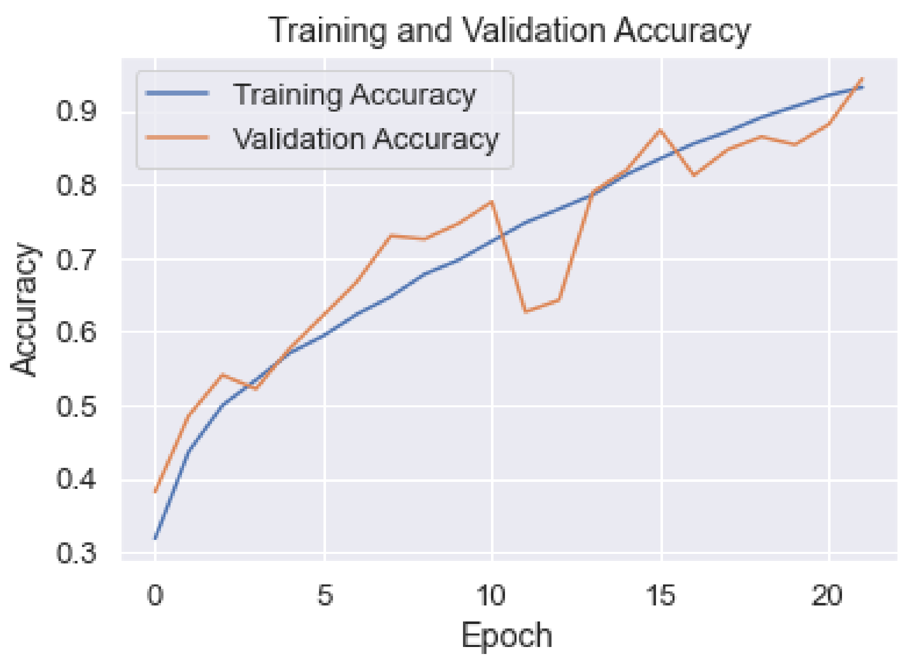
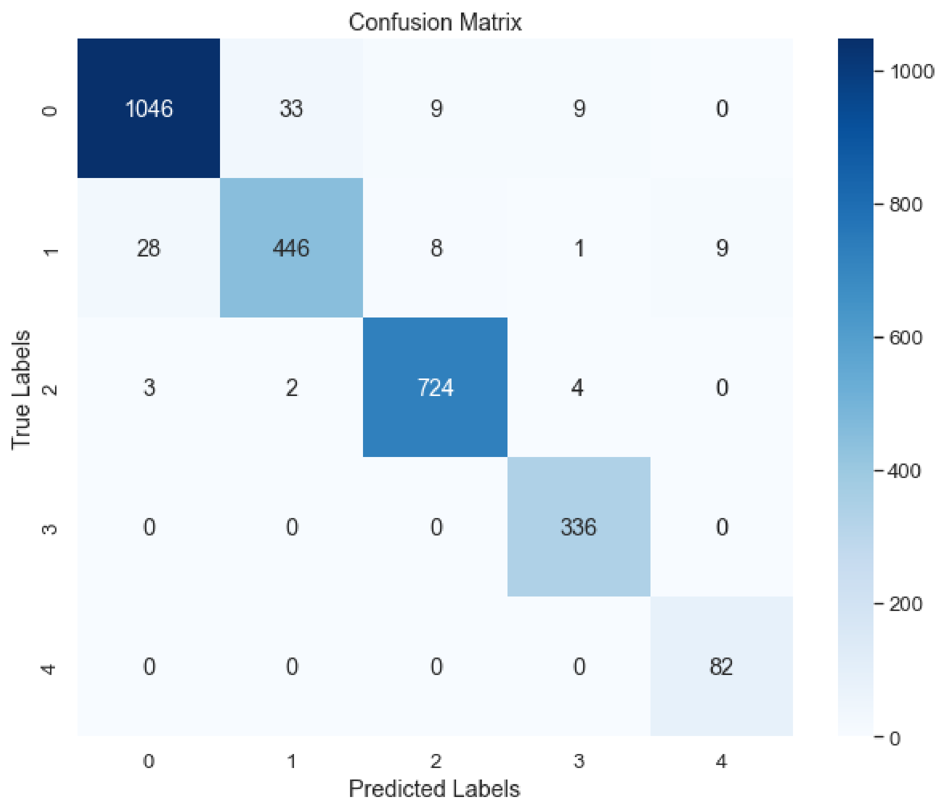


Image 18: Confusion Matrix for the Xception Model



True \ Predicted	0	1	2	3	4
0	1046	33	9	9	0
1	28	446	8	1	9
2	3	2	724	4	0
3	0	0	0	336	0
4	0	0	0	0	82

<u>**Development of a website for Knee Osteoarthritis Prediction**</u>

Our team successfully built a user-friendly website utilizing Flask for knee osteoarthritis (OA) prediction through X-ray image analysis. This platform simplifies the diagnostic process by allowing healthcare providers to upload knee X-ray images and receive predictions regarding the severity of knee osteoarthritis. This innovative tool streamlines the assessment process, offering quick and accurate insights into patients' conditions. With its intuitive interface, the website is poised to greatly assist healthcare professionals in hospitals and clinics. By simply uploading knee X-ray images, doctors can efficiently obtain valuable information for diagnosis and treatment planning. Overall, our Flask-based website represents a significant advancement in healthcare technology, offering a practical and accessible solution for knee osteoarthritis diagnosis.

Image 19: Screenshot of website for classification

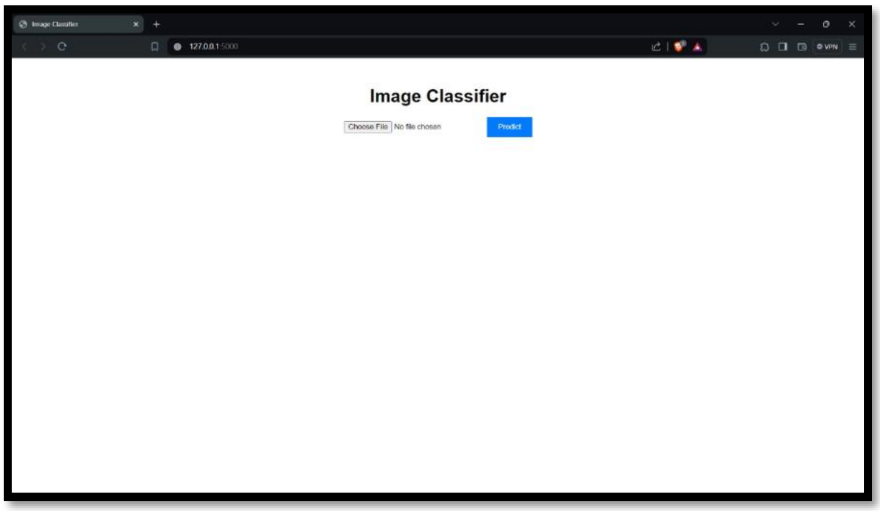


Image 20

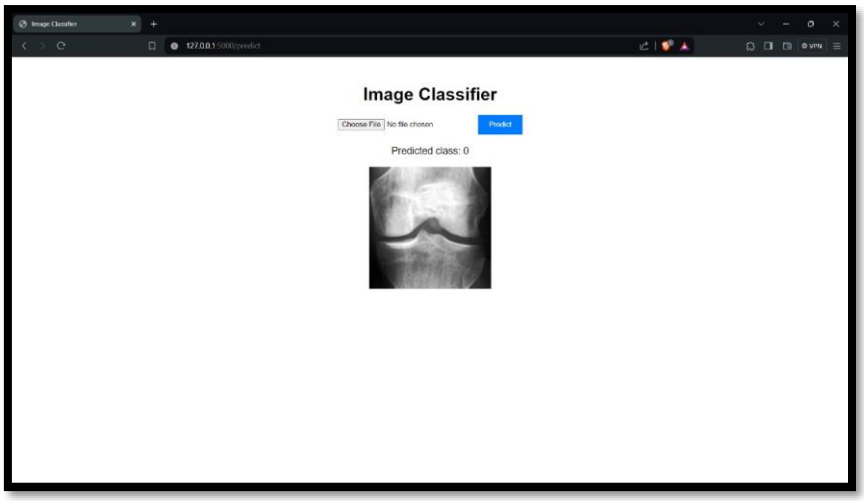
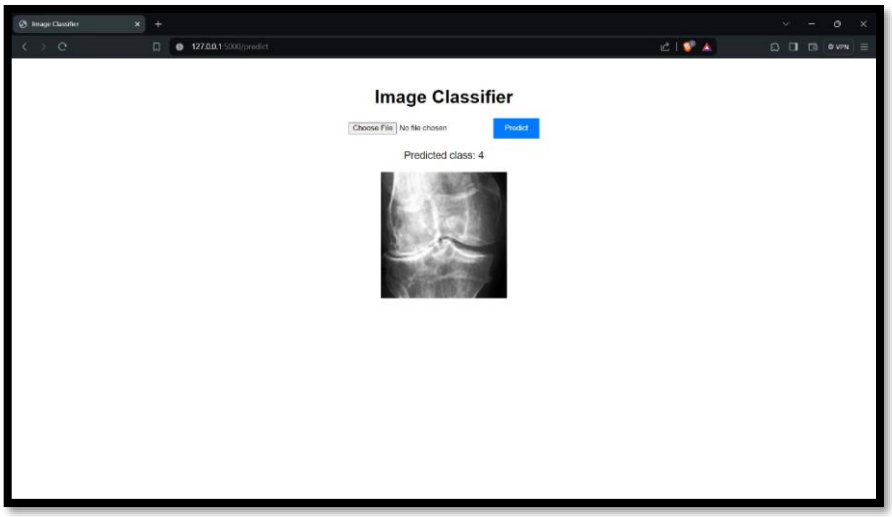


Image 21



Images 20 and 21 demonstrate the successful upload and accurate classification of knee X-ray images.

Conclusion

This project successfully developed an automated system for classifying knee osteoarthritis severity using X-ray images. We explored the effectiveness of two powerful deep learning models: VGG19 and Xception. Both models were enhanced with customized transfer learning strategies, class weights to address imbalance, and tailored classification heads.

Careful analysis of the classification reports and confusion matrices indicates that both models achieved promising results. The Xception model demonstrated a slight edge in performance, likely due to its computationally efficient depth wise separable convolutions. Its reduced complexity contributed to faster training and potentially better resistance to overfitting.

This work highlights the potential of deep learning models for aiding in the diagnosis of knee osteoarthritis. The developed system could assist medical professionals by providing them with a reliable and automated tool for assessing X-ray images.