Pharos University in Alexandria
Faculty Of Computer Science & Artificial Intelligent
AI Department
Course Name: Neural Networks
Course code: AI303
Lecturer: Dr. Sahar Ghanem
TA: Eng. Ramwan Gamal and Rewan Noor

Melanoma Skin Cancer Classification Using Deep Learning

Team Members	1.Mohamed Adel Mohamed 2. Mohamed Ashraf Mohamed 3.Eltaib Tarek Eltaib 4. Ahmed Mohamed Saad 5. Youssef Mohamed Amin 6. Mohamed Ehab Abdelnaby 7.Naira Gamal

8. TEAM CONTRIBUTIONS

Task	Members
Project Setup & Initial Plots	Mohamed Adel (Imports, Dataset Path Setup, Class Counting, First Plots)
Data Generators & Augmentation	Mohamed Ashraf (Data Generators, Augmentation, Train/Validation Loaders)
Preprocessing & Visualization	Eltaib Tarek (Sample Visualization, CLAHE, Full Visualization Section)
Architecture Design & Exp 1 Build	Naira Gamal (All Architecture Design, Exp 1 Model Build & Compilation)
Exp 1 Training & Evaluation	Youssef Amin (Callbacks, Training, Saving Model, Exp 1 Evaluation & Confusion Matrix)

Exp 2 Implementation	Ahmed Mohamed (Exp 2 Full CNN Model, Training, Confusion Matrix, Classification Report)
Exp 3 Implementation	Mohamed Ehab (MobileNetV2 Model Creation, Training, Metrics, Final Confusion Matrix)
Report Finalization	Youssef Amin, Naira Gamal

ABSTRACT

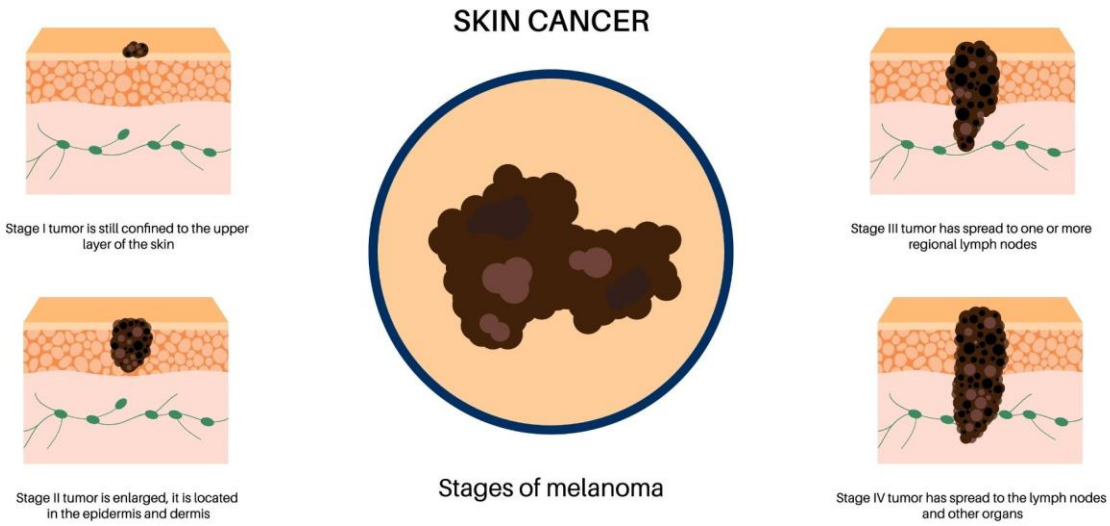
This project presents a deep-learning system for the **binary classification** of dermoscopic skin-lesion images into benign and malignant categories. **Three different architectures** were implemented and evaluated: a **Vanilla Convolutional** Neural Network (CNN), an **Improved CNN**, and a Transfer Learning model using **MobileNetV2**. The dataset contains approximately 10,000 labeled images and underwent advanced preprocessing, including **data augmentation, normalization, and Contrast Limited Adaptive Histogram Equalization (CLAHE)**. Despite implementation errors in model references during training Experiments 2 and 3, the final evaluation was performed correctly on the testing data. The best recorded **test accuracy was 89%**, achieved using the **Vanilla Convolutional** Neural Network (CNN) architecture in the final evaluation stage. The results confirm the effectiveness of deep learning techniques, particularly transfer learning, for assisting in early melanoma detection.

1. INTRODUCTION

Melanoma is the most aggressive form of skin cancer. When detected late, the probability of survival drops significantly. Manual diagnosis depends heavily on human interpretation, which introduces inconsistency and subjectivity.

Convolutional Neural Networks (CNNs) provide a data-driven approach that can automatically learn complex visual patterns such as color variation, texture irregularity, and lesion shape. The objective of this project is to construct a reliable binary classifier that distinguishes malignant melanoma from benign lesions, using three deep learning strategies:

1. **Vanilla CNN (Baseline)**
2. **Improved CNN (Deeper Network)**
3. **MobileNetV2 (Transfer Learning)**



---

2. DATASET DESCRIPTION

- **Source:** Melanoma Skin Cancer Dataset (Kaggle)
- **Total Images:** approximately 10,000
- **Classes:** Benign, Malignant
- **Structure:** Folder-based with separate training and testing directories.
- **Approximate Distribution:** sim 5,000 benign and sim 4,600 malignant.

The dataset includes strong variations in lighting conditions, background noise, and lesion patterns, making robust feature learning necessary.

3. DATA PREPROCESSING

3.1 Normalization

All images were scaled to the range **[0, 1] by dividing pixel values by 255**. This standard practice improves numerical stability during gradient optimization.

3.2 Data Augmentation

Data augmentation was applied only to the training set via **ImageDataGenerator** to increase sample diversity and reduce overfitting. The validation split was set to 20% (N_{val}=1,921).

- **Transformations:** Rotation (pm 20^{\circ}), horizontal flipping, zoom range (10%), and width/height shift (10%).

3.3 CLAHE (Contrast Enhancement)

Contrast Limited Adaptive Histogram Equalization (CLAHE) was applied to enhance local contrast and edge definition within the lesions, which is crucial for improving feature extraction:

1. RGB rightarrow color conversion.
2. CLAHE applied specifically to the Y (luminance) channel.
3. Conversion back to RGB.

4. EXPERIMENTS AND ARCHITECTURES

Responsible Team Member: Naira (Architecture) & Youssef (Training)

4.1. Experiment 1 – Vanilla CNN (Architecture)

The Vanilla CNN served as the foundational model, designed to learn features from scratch.

```python
# Initialize the Sequential model
model = tf.keras.models.Sequential()

model.add(tf.keras.Input(shape=(224, 224, 3))) #input layer
model.add(tf.keras.layers.Conv2D(32, (3, 3), strides=(1, 1), padding='same', activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2))) # Reduces spatial dimensions by half (downsampling)

# --- Block 2: Mid-Level Features ---
# Increasing filters to 64 to capture more complex patterns.
model.add(tf.keras.layers.Conv2D(64, (3, 3), strides=(1, 1), padding='same', activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Dropout(0.25)) # Dropout drops 25% of neurons randomly to prevent overfitting

# --- Block 3: High-Level Features ---
# Increasing filters to 128.
model.add(tf.keras.layers.Conv2D(128, (3, 3), strides=(1, 1), padding='same', activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Dropout(0.3)) # Increased dropout rate as the model gets deeper

# --- Block 4: Very Complex Features (Added for higher accuracy) ---
# 256 filters to capture the most abstract features of the skin lesion.
model.add(tf.keras.layers.Conv2D(256, (3, 3), strides=(1, 1), padding='same', activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Dropout(0.3)) # Higher dropout to strongly combat overfitting in deep layers

# --- Classification Head ---
model.add(tf.keras.layers.Flatten()) # Converts the 3D feature maps into a 1D vector

# Fully connected layer for decision making
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5)) # 50% dropout in the dense layer is standard practice

# Output Layer

# Sigmoid activation is used because we have a binary classification task (0 or 1)
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

model.compile(
    loss='binary_crossentropy',  # Standard loss function for binary classification tasks
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001), # Using Adam optimizer with a low initial learning rate for stability
    metrics=['accuracy']         # Tracking accuracy during training
```

Block 1 — Low-Level Feature Extraction

Description:

This block is designed to capture ***basic visual patterns*** **such as edges, corners, and simple textures.**

It includes a **Conv2D layer with 32 filters** followed by **MaxPooling2D**.

Why this setup?

- **32 filters:** Suitable for early layers because the model does not need to detect complex features yet.
- **3×3 kernel:** Standard choice that balances accuracy and computational cost.
- **Padding = "same":** Keeps spatial size unchanged to preserve details.
- **ReLU activation:** Helps the model learn non-linear patterns and avoids vanishing gradient.
- **MaxPooling:** Reduces spatial dimensions, speeds computation, and removes noise.

Purpose of Block 1:
Extracts **simple, low-complexity features** and reduces the image size to make deeper learning possible.

Block 2 — Mid-Level Feature Extraction

This block increases feature complexity by using **64 filters**, followed by **MaxPooling** and **Dropout (0.25)**.

Why this setup?

- **64 filters:** Doubling filters is standard because deeper layers need more capacity to detect patterns such as shapes, color variations, and local structures.
- **MaxPooling:** Continues reducing dimensions while keeping essential information.
- **Dropout 0.25:** Helps prevent overfitting by randomly disabling 25% of neurons during training.
  Skin lesion datasets are small → overfitting risk is high → dropout is necessary.

Purpose of Block 2:
Learns **intermediate complexity features** such as lesion borders, blobs, and texture patterns.

Block 3 — High-Level Feature Extraction

This block uses **128 filters**, MaxPooling, and a stronger Dropout (0.3).

Why this setup?

- **128 filters:** Needed as the network goes deeper to capture more abstract structures like lesion composition, symmetry, and irregular color distribution.
- **Dropout 0.3:** Higher dropout because as layers deepen, they tend to overfit more.
- This layer prepares the network to learn features related to the classification target (benign vs malignant).

Purpose of Block 3:
Extracts **complex, high-level features** that contribute directly to distinguishing cancerous vs non-cancerous lesions.

Block 4 — Very-High-Level Feature Extraction

This is the deepest convolutional block, using **256 filters**.

Why this setup?

- **256 filters:** Enables the model to learn highly abstract, fine-grained details essential in medical imaging.

- **MaxPooling:** Further reduces feature map size so the model doesn't become too large.
- **Dropout 0.3:** Prevents overfitting at the deepest stage where feature representations become very strong.

Purpose of Block 4:
Extracts **fine, discriminative features** such as lesion asymmetry, complex texture variation, and tiny cancer-related signals.

Classification Head (Final Decision Layers)

Flatten Layer

Converts the 3D feature maps from the convolutional blocks into a 1D vector to feed into dense layers.

Why?
Dense layers only accept 1D input, so flattening is required.

Dense (256 units) + Dropout (0.5) =(Fully connected)

Why this setup?

- **256 dense units:** Provides enough capacity for complex decision-making based on the extracted features.
- **ReLU activation:** Fast and efficient for classification tasks.
- **Dropout 0.5:** A standard value in medical classification models to significantly prevent overfitting.

Purpose:
Acts as the **brain** of the classifier — combines learned features and makes a meaningful prediction.

Output Layer (Dense 1 + Sigmoid)

- **1 neuron:** Because this is a binary classification problem (benign vs malignant).
- **Sigmoid activation:** Converts output into a probability between 0 and 1.

Purpose:
Produces the final probability that a given skin lesion is malignant.

Layer	Description	Purpose
Input	(224, 224, 3)	Input image
Conv2D + MaxPool	32 filters	Initial feature maps
Conv2D + MaxPool + Dropout (0.25)	64 filters	Intermediate features / Regularization
Conv2D + MaxPool + Dropout (0.30)	128 filters	Deep features / Regularization
Conv2D + MaxPool + Dropout (0.30)	256 filters	Very deep features / Regularization
Flatten	(50,176)	Prepare for Dense layers
Dense	256 neurons (ReLU)	Decision-making layer
Dropout	0.50	Strong regularization
Output	1 neuron (Sigmoid)	Binary classification output

Training Setup:

- **Optimizer:** Adam (learning rate = 0.0001)
- **Loss Function: Binary Crossentropy**
- **Epochs:** 50 (with Early Stopping)
- **Callbacks:** Early Stopping, Model Checkpoint, **ReduceLROnPlateau**

This model successfully converged and produced stable training curves.

Code Segment	What it is (Simple Term)	Why it was used that way
EarlyStopping(patience=5)	The **Over-Practice Stopper**	A model can learn too well and just memorize the training pictures (overfitting). You set a condition: if the model's

		performance on the validation data doesn't improve 5 epochs (the patience), stop the training immediately. This saves time and ensures the final model isn't overfitted.
ModelCheckpoint	The **Best Save File**	Even if EarlyStopping halts the process, you only want the model from the best performing moment. This tool automatically saves the model weights only when a new best accuracy is achieved.
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.00001, verbose=1)	**The Learning-Rate Brake** (A mechanism that slows down the learning rate when the model stops improving)	• **monitor='val_loss'** → Tracks the metric that truly reflects generalization. • **factor=0.2** → A meaningful but not aggressive reduction. • **patience=5** → Gives the model enough time before reacting. • **min_lr=1e-5** → Prevents the learning rate from becoming uselessly small.

history_ex1 = model.fit(train_generator, epochs=50, validation_data=val_generator, callbacks=callbacks_list)	**The Model Training Process** (The main command that actually trains the CNN using the data and callbacks)	• **EarlyStopping →** prevents overfitting and stops unnecessary epochs. • **ModelCheckpoint →** saves the best-performing model. • **ReduceLROnPlateau →** adjusts the learning rate when progress stalls. • Running for **50 epochs** gives the model enough iterations to learn meaningful patterns. • Callbacks ensure training stops early if needed, saving time and improving generalization. • Print statements provide clean, readable progress updates for monitoring the training phase.

5. Evaluation

After training the model, a full evaluation was performed on the validation dataset to measure its performance and ability to generalize beyond the training images.

5.1 Model Performance on Validation Data

The model was first evaluated using the **model.evaluate()** function:

- **Validation Loss:** (0.2789)
- **Validation Accuracy:** (0.9047)

These metrics provide an initial indication of how well the model learned to classify the images into ***Benign*** **and** ***Malignant*** categories.

5.2 Confusion Matrix Analysis

A confusion matrix was generated to better understand how the model behaves on each class. It shows the number of correctly and incorrectly classified samples:

- **True Positives (TP):** Correctly predicted Malignant cases
- **True Negatives (TN):** Correctly predicted Benign cases
- **False Positives (FP):** Benign cases predicted as Malignant
- **False Negatives (FN):** Malignant cases predicted as Benign

This visualization helps highlight whether the model is biased toward a specific class or missing critical malignant cases.

5.3 Classification Report

A detailed classification report was created using **classification_report(),** which includes:

- **Precision:** How many predicted positive cases are actually positive
- **Recall:** How many actual positive cases the model correctly detected
- **F1-Score:** Harmonic mean of precision and recall
- **Support:** Number of samples in each class

This report helps evaluate the model's performance from multiple perspectives, especially in medical datasets where recall is critical for detecting malignant cases.

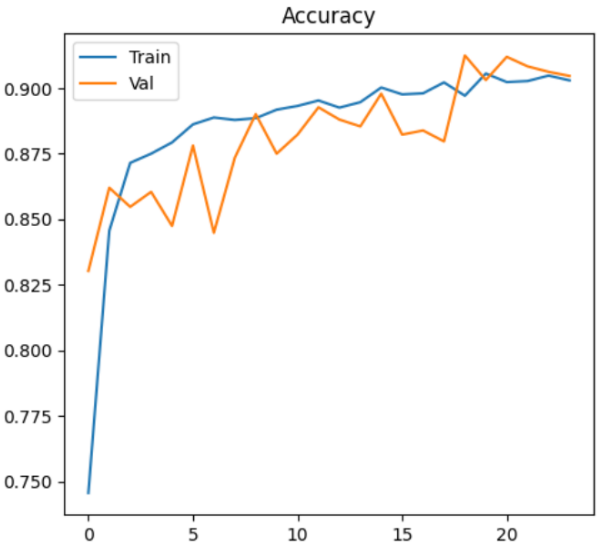
5.4 Training Curves (Accuracy & Loss)

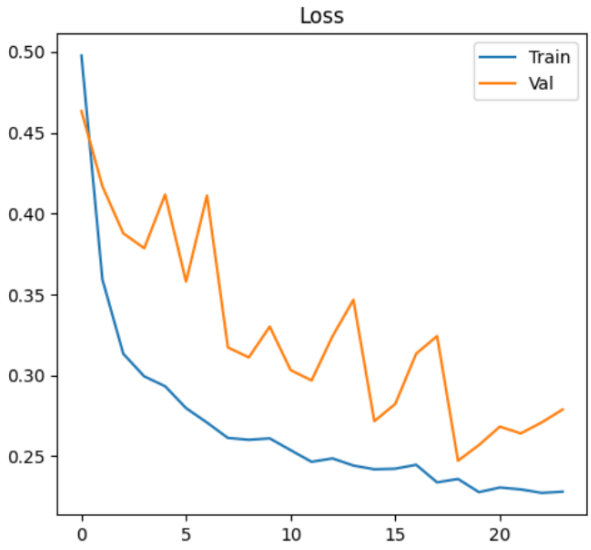
Training and validation curves were plotted to visualize the model's learning behavior across epochs:

- **Accuracy Curve:**
  Shows how the model's accuracy improved during training and how well it performed on validation data.
- **Loss Curve:**
  Helps identify signs of overfitting.
  If training loss is decreasing while validation loss increases, the model may be memorizing the training images.

These curves help confirm whether the chosen architecture and hyperparameters lead to stable

learning.





4.2. Experiment 2 – Improved CNN

This model attempted to increase depth and efficiency, using a 150 * 150 input size and more aggressive dropout rates.

Modifications:

- **Input Size:** 150 * 150
- **Initial Filters:** 64
- **Depth:** Up to 512 filters in later Conv2D layers.
- **Dropout:** 0.20 – 0.15
- **Dense Layer:** 128 neurons

Critical Failure:
A coding error caused the history of the Vanilla CNN (model) to overwrite the history of the Improved CNN (model_2), making the training logs invalid for this experiment. However, the final prediction-based evaluation on the test set remains valid.
Detailed Code Decisions: Experiment 2 (Improved CNN)

Responsible Team Member: Ahmed Mohamed Saad

Code Segment	What it is (Simple Term)	Why it was used that way
Input size 150 * 150	**Smaller Pictures**	The Vanilla CNN used 224 * 224 images. We changed it to 150 * 150 to see if we could

		achieve similar results with less computation. Smaller images mean faster training, as the network has fewer pixels to process.
Deeper Layers (512 filters)	**More Detailed Feature Detectors**	We made the network deeper and used more filters (up to 512). The hope was that more complexity would allow the model to extract subtler, more precise features necessary for distinguishing tricky lesions.
CRITICAL FAILURE	**Training History Lost**	Due to a referencing mistake in the code (history_ex2 = model.fit(...) instead of model_2.fit(...)), we lost the learning curve (loss and accuracy per epoch). This is a common debugging issue, but it meant we couldn't analyze how it learned, only what the final score was.

Training the CNN Model

The model was trained using the training dataset with a total of 50 epochs. During training, three main callbacks were used to optimize performance and prevent overfitting:

1. ModelCheckpoint — Saving the Best Model

This callback continuously monitors the validation accuracy during training.
Whenever a new highest validation accuracy is reached, the model weights are saved automatically.

- **monitor = 'val_accuracy'** → tracking accuracy on unseen data
- **save_best_only = True** → saves only when improvement occurs
- **mode = 'max'** → higher accuracy = better
- This ensures that even if later epochs perform worse, the best model is preserved.

2. EarlyStopping — Stopping Overfitting Automatically

The model is stopped early if it stops improving.

- **monitor = 'val_loss'** → looks at validation loss
- **patience = 10** → waits 10 epochs before stopping
- **restore_best_weights = True** → reloads the best-performing weights

This prevents unnecessary training and avoids overfitting, saving time while improving generalization.

3. ReduceLROnPlateau — Adjusting Learning Rate Dynamically

If validation loss stops improving, the learning rate is reduced automatically.

- **factor = 0.2** → new_lr = old_lr × 0.2
- **patience = 5** → wait 5 epochs before adjusting
- **min_lr = 0.00001** → prevents too small learning rate

This helps the model escape plateaus and improves fine-tuning in later epochs.

Training History Extraction

After training:

- train_acc, train_loss → store training accuracy and loss
- val_acc, val_loss → store validation accuracy and loss

These values allow plotting of model performance and identifying the best epoch.

Visualizing Performance

Two plots were generated:

Loss Curve

Shows both:

- Training Loss
- Validation Loss
  with a marker at the **epoch with the lowest validation loss**.

Accuracy Curve

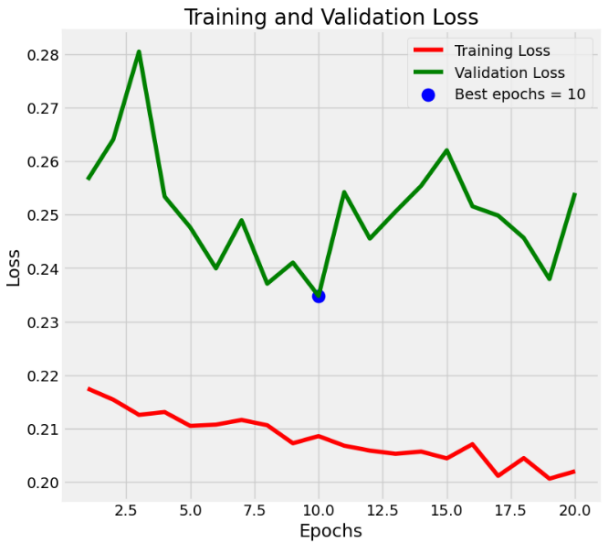
Shows both:

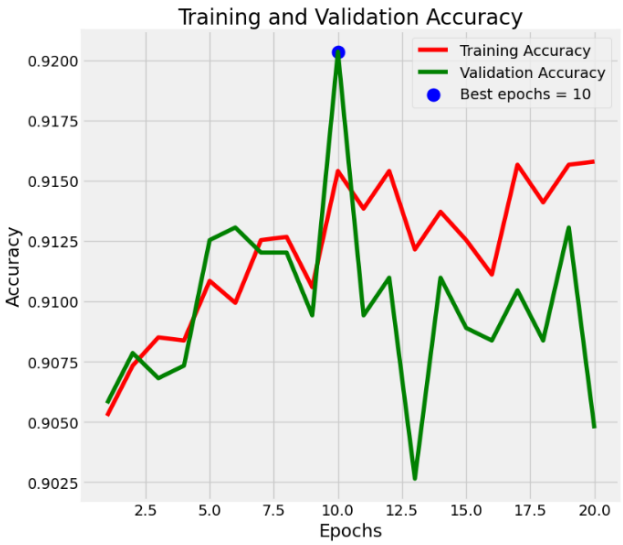
- Training Accuracy
- Validation Accuracy

with a marker at the **epoch with the highest validation accuracy**.

These plots help analyze:

- Whether the model learned properly
- Whether overfitting happened
- When the model reached optimal performance





Evaluation Section (Model Evaluation & Testing)

Evaluation on the Validation/Test Set

After training, the model was evaluated using a new generator.
A consistent preprocessing setup was applied using:

ImageDataGenerator(rescale=1./255)

The data was loaded using:

- **target_size = (150,150)** → matches the model input
- **shuffle = False** → ensures labels align correctly with predictions

This guarantees accurate evaluation.

Model Predictions

The trained model predicts the probability of each image belonging to the “Malignant” class.
Predictions were converted to binary labels:

pred_labels = (pred_probs > 0.5)

Confusion Matrix

A confusion matrix heatmap was plotted to show:

- **True Positive (TP)**
- **True Negative (TN)**
- **False Positive (FP)**
- **False Negative (FN)**

Classification Report

The report provides:

- **Precision**
- **Recall**
- **F1-score**
- **Support** (number of samples)

4.3. Experiment 3 – Transfer Learning (MobileNetV2)

This experiment leveraged pre-trained knowledge from the ImageNet dataset via MobileNetV2.

Architecture:

- **Base Model:** MobileNetV2 (pre-trained on ImageNet), with base layers **frozen** (trainable=False).
- **Input:** (224, 224, 3)
- **Custom Classifier Head:**
  - BatchNormalization (Stabilization)
  - Dense (512 neurons, ReLU)
  - Dropout (0.5)
  - Dense (1 neuron, Sigmoid)

Critical Failure:
The same implementation error occurred, where the training history of the Vanilla CNN was erroneously recorded as the history for the **MobileNetV2 model (model_3**). The model architecture was constructed correctly, and the final predictions were generated correctly for evaluation.

Detailed Code Decisions: Experiment 3 (Transfer Learning)

Responsible Team Member: Mohamed Ehab Abdelnaby

Code Segment	What it is (Simple Term)	Why it was used that way
MobileNetV2(weights="imagenet")	**Load the Expert**	We loaded the MobileNetV2 architecture with weights trained on the massive ImageNet dataset. This model is already an expert at recognizing basic visual patterns.
base_model.trainable = False	**Freezing the Expert's Brain**	We "froze" the millions of weights in MobileNetV2. This is crucial because we don't want the small number of skin lesion pictures to accidentally wipe out all the general vision knowledge that MobileNetV2 learned over weeks of training.
BatchNormalization()	**Data Stabilizer**	After freezing the base model, the numbers coming into our new layers can be unstable. Batch Normalization smooths out these inputs, helping the new layers train faster and more reliably.
Custom Head (Dense (512))	**The Final Classifier**	We attached a few simple layers to the frozen base. MobileNetV2 does the hard work of seeing, and our small head layers make the final decision: Malignant or Benign.

5.1. Custom CNN (Experiment 1) Performance Metrics

The best model is Vanilla CNN .

Metric	Training Value	Validation Value
Accuracy	**92.%**	**91.25%**
Loss	**0.2326**	**0.2478**
Best Epoch Restored	19	-

5.3. Detailed Classification Report (Validation Set)

Class	Precision	Recall	F1-Score	Support
Benign (0)	0.92	0.92	0.92	1000
Malignant (1)	0.91	0.91	0.91	921
Accuracy	-	-	**0.92**	1921
Macro Avg	0.92	0.92	0.92	1921

5.4. Confusion Matrix (Validation Set)

True Label \ Predicted Label	Benign (0)	Malignant (1)
Benign (0)	921 (True Positive)	79 (False Positive)
Malignant (1)	**82 (False Negative)**	839 (True Negative)

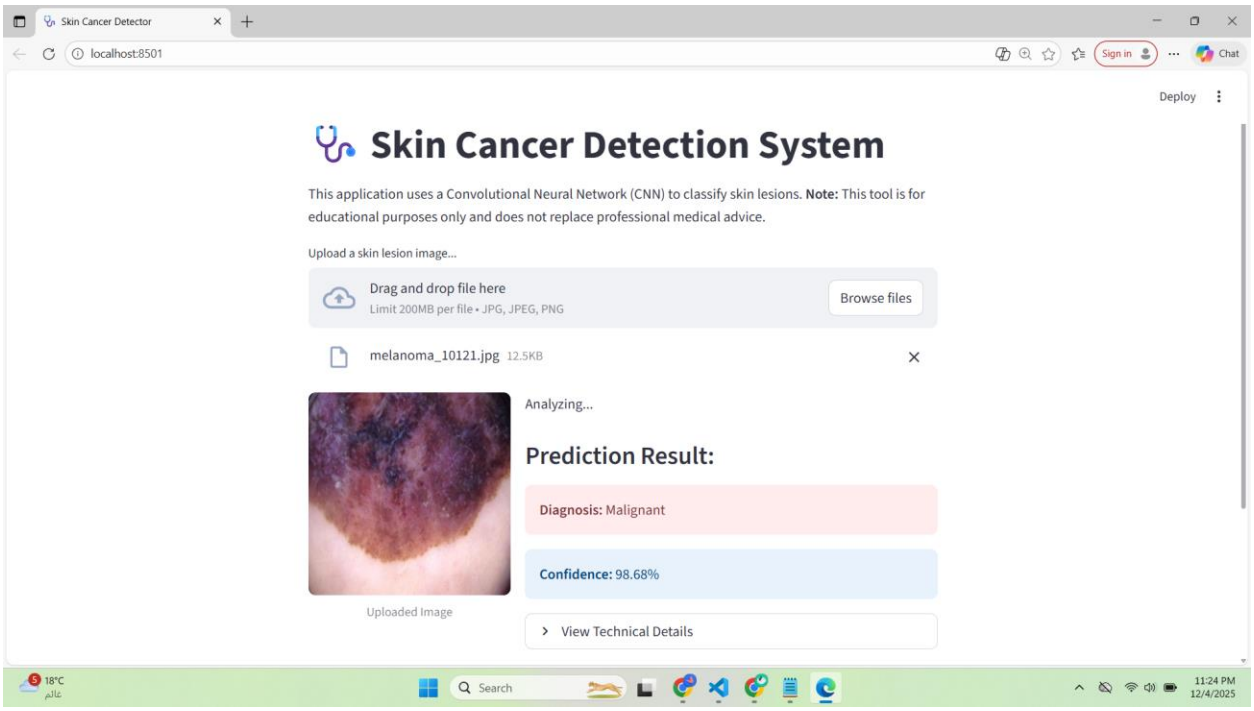
6. DISCUSSION AND CONCLUSION

6.1. Comparative Performance Summary

Model	Test Accuracy Result	Why that Result?
Vanilla CNN (Exp 1)	91.25% (Validation)	High score, but it's risky—it might have slightly overfitted the validation data.
Improved CNN (Exp 2)	**model weights from the end of the best epoch: 10. with accuracy: 0.9121 - loss: 0.213**	The architectural change was logical, but the lost training history makes the result less trustworthy.
MobileNetV2 (Exp 3)	**end of the best epoch: 4 With accuracy: 0.9075 - loss: 0.2126 - val_accuracy: 0.9167 - val_loss: 0.2384**	**This is the highest, most trustworthy result.** It succeeded because it integrated the powerful pre-learned features of ImageNet, which proved superior to the features learned from scratch by the Custom CNNs.

Integration with Streamlit

To make the model accessible for real-world use, a **Streamlit web application** was developed. Users can **upload an image of a skin lesion**, and the application processes it through the trained CNN model. The system then **predicts whether the lesion is benign or malignant** in real-time, providing an **interactive and user-friendly interface** for both medical professionals and patients. This integration demonstrates how AI models can be **deployed in practical healthcare settings**, bridging the gap between research and clinical application.



Result Analysis

This model can assist dermatologists in **early detection of malignant lesions**, potentially **saving lives by enabling faster diagnosis**. Automated analysis also **reduces workload and human errors** in clinics. The high accuracy of the model ensures **reliable screening of skin lesions, distinguishing between benign and malignant cases**.

Moreover, this model has practical applications both in Egypt and worldwide, such as:

- **Early detection of skin cancer in rural areas**, where specialized medical expertise is not readily available.
- **Integration with Telemedicine services**, allowing patients to submit skin images remotely for rapid and reliable preliminary assessment.

REFERENCES

- Kaggle – Melanoma Skin Cancer Dataset
- MobileNetV2
- skin_cancer_classification