Real vs Fake Face Classification Using VGG16 CNN on 140K Face Dataset

A Deep Learning-Based Approach for Binary Image Classification

Course: Deep Learning – Spring 2024/2025

Course instructor: Dr. Ala’a Al-Habashna

Leen Shareef Saleh - 22110090

Table of Contents

[Problem statement 3](#_Toc200413052)

[Research on the Neural Networks and architectures 4](#_Toc200413053)

[Neural Networks used for the problem 4](#_Toc200413054)

[Simple Convolutional Neural Network (SimpleCNN) 5](#_Toc200413055)

[Modern architectures 5](#_Toc200413056)

[VGGNet (VGG16 / VGG19) 6](#_Toc200413057)

[ResNet (Residual Network) 6](#_Toc200413058)

[Inception Networks (GoogLeNet, Inception-v3) 6](#_Toc200413059)

[MobileNet / EfficientNet 7](#_Toc200413060)

[DenseNet (Densely Connected Convolutional Networks) 7](#_Toc200413061)

[Modern architectures comparison 8](#_Toc200413062)

[Models’ development and training 9](#_Toc200413063)

[Dataset 9](#_Toc200413064)

[Dataset Preparation 9](#_Toc200413065)

[Training and validation 10](#_Toc200413066)

[3-way Splitting 10](#_Toc200413067)

[hyperparameters for each architecture and their values 10](#_Toc200413068)

[Combinations of the hyperparameter values and corresponding performance achieved 11](#_Toc200413069)

[learning curve (training and validation performance vs epochs for each model (best) 13](#_Toc200413070)

[SimpleNN 13](#_Toc200413071)

[GVVnet16 13](#_Toc200413072)

[RESnet18 14](#_Toc200413073)

[DenseNet121 14](#_Toc200413074)

[Epochs vr Acurracy for the best combo of the 4 models 15](#_Toc200413075)

[Datasize vs Acurracy 16](#_Toc200413076)

[Models’ testing and evaluation 17](#_Toc200413077)

[Testing 17](#_Toc200413078)

[Evaluation Metrics Chosen 17](#_Toc200413079)

[Values for the evaluation metrics achieved on the test set for the best model from each architecture. 18](#_Toc200413080)

[SimpleNN 18](#_Toc200413081)

[VGGnet16 19](#_Toc200413082)

[RESnet18 20](#_Toc200413083)

[denseNet121 21](#_Toc200413084)

[Over/under-fitting assessment 22](#_Toc200413085)

[SimpleNN 22](#_Toc200413086)

[VGGnet16 22](#_Toc200413087)

[RESnet18 23](#_Toc200413088)

[denseNet121 23](#_Toc200413089)

[Results analysis 24](#_Toc200413090)

[Effectiveness assessment 24](#_Toc200413091)

[Interface development 25](#_Toc200413092)

[Critical evaluation of models 26](#_Toc200413093)

**Developing a Deep-Learning system for a Computer Vision Application**

# Problem statement

In an era where digital media is easily manipulated, distinguishing between real and synthetic human faces has become both a technical challenge and a societal concern.[[1]](#footnote-1) The rise of AI-generated faces - such as those produced by GANs[[2]](#footnote-2) (Generative Adversarial Networks) - poses risks ranging from misinformation to identity fraud. The aim of this project is to build a deep learning-based image classification system capable of identifying whether a given facial image is real (captured from actual human faces) or fake (AI-generated).

This is a **binary classification problem**, with two clearly defined classes:

* **Class 0 - Real Faces:** Genuine facial images of real people.
* **Class 1 - Fake Faces:** AI-generated faces (from tools like ThisPersonDoesNotExist).

The input to the system is a facial image, and the output is a predicted label (0 or 1) indicating whether the face is real or fake. The goal is to train a robust model that generalizes well to unseen images and can achieve high classification accuracy despite subtle differences between classes.

Solving this problem has practical applications in digital forensics, social media integrity, and cybersecurity.

|  |  |
| --- | --- |
| fake | A person with short hair holding her head  AI-generated content may be incorrect.  Real |

Link to the dataset: <https://www.kaggle.com/datasets/xhlulu/140k-real-and-fake-faces>

# Research on the Neural Networks and architectures

## Neural Networks used for the problem

The task of distinguishing between real and AI-generated human faces is a visual pattern recognition problem - one that requires a model capable of analyzing and interpreting image data at both low and high levels. In deep learning, the most effective models for such tasks are **Convolutional Neural Networks (CNNs)**.[[3]](#footnote-3)

CNNs are specially designed for processing grid-like data structures such as images. Unlike traditional fully connected networks, CNNs apply **localized filters (kernels)** to capture spatial relationships between pixels.[[4]](#footnote-4) These filters slide over the image, detecting features such as edges, corners, and textures, which are later combined to form more abstract representations - such as facial contours or lighting patterns. This makes CNNs uniquely powerful for image classification tasks where spatial coherence matters.[[5]](#footnote-5)

In our case, the visual differences between real and synthetic faces can be extremely subtle. Fake faces generated by advanced models like **StyleGAN[[6]](#footnote-6)** often look natural to the human eye, but they still exhibit artifacts - irregularities in texture, symmetry, or lighting. These nuances are often too complex to capture with shallow or manually engineered features. However, a well-trained CNN can learn to detect them automatically through its hierarchical layers of abstraction.

Here’s how CNNs operate in this context:[[7]](#footnote-7)

* Convolutional layers, which apply learned filters to detect local features such as edges, curves, and textures. These features become more abstract as the network depth increases.
* Activation functions like the Rectified Linear Unit (ReLU) are applied after each convolution. ReLU introduces non-linearity, allowing the network to learn complex, non-linear relationships between input features.
* Pooling layers, typically max pooling, reduce the spatial dimensions of the feature maps. This helps retain the most prominent features while reducing computational load and overfitting.
* The output of the convolutional and pooling stages is flattened and passed through one or more fully connected layers, which function similarly to traditional neural networks. These layers combine the learned features to produce a final prediction. For binary classification problems like this one, a sigmoid activation function is typically used in the output layer.

This layered approach mirrors the multilayer perceptron (MLP) structure but with a key difference: CNNs **preserve the spatial structure of the input**, rather than flattening it immediately. This preservation is crucial in our problem, where pixel position and local structure contain valuable clues about authenticity.

### Simple Convolutional Neural Network (SimpleCNN)

SimpleCNN is a custom-built convolutional neural network designed specifically for this binary image classification task. Unlike large-scale pretrained models, SimpleCNN is constructed from scratch using a few convolutional and fully connected layers. This architecture offers a balance between simplicity and sufficient depth to learn meaningful visual patterns.

SimpleCNN is composed of:

* Two convolutional layers with 3×3 filters and ReLU activations
* Max pooling layers to reduce spatial dimensions
* A fully connected dense layer followed by dropout to prevent overfitting
* A sigmoid-activated output layer to produce a probability score for binary classification

This model is particularly useful as a baseline, offering fast training time and easy interpretability. While its performance may not match larger pretrained models, it helps establish a foundational comparison point for evaluating model complexity vs. accuracy.

* **Strength**: Lightweight, fast to train, easy to interpret, and works well on smaller datasets.
* **Limitation**: May underperform on complex visual patterns compared to deeper architectures like ResNet or DenseNet.

Note that VGGnet16/DENSnet121/RESnet18 – were also used, but their explaination is provided by the modern architectures section

## Modern architectures

Convolutional Neural Networks (CNNs) form the foundation of most modern deep learning solutions for image classification problems. However, over the years, researchers have developed several architectural variants that improve performance, depth, computational efficiency, and generalization capabilities.[[8]](#footnote-8)

In this section, we review four widely adopted CNN architectures that are especially relevant for our task of binary classification between real and AI-generated facial images.

### VGGNet (VGG16 / VGG19)[[9]](#footnote-9)

VGGNet is one of the most influential CNN architectures and is often used as a baseline in image classification tasks. It consists of multiple layers of small **3×3 convolutional filters**, stacked one after another with ReLU activations and occasional max-pooling layers. Its power lies in its simplicity and depth, which allows the network to extract increasingly complex features from images.

In the context of our problem, VGG is well-suited for learning the **fine-grained textural differences** between real and synthetic faces. The deep, layered design enables it to capture patterns such as lighting irregularities, unnatural skin smoothness, or inconsistencies in background elements - common in AI-generated faces.

* **Strength:** Simple to implement and interpret; deep enough to learn subtle patterns.
* **Limitation:** High memory and computational demands during training and inference.

### ResNet (Residual Network)[[10]](#footnote-10)

ResNet introduced the concept of **residual connections**, which allow the model to learn identity functions by skipping layers. This solved the degradation problem observed in very deep networks, enabling successful training of models with 50, 101, or even 152 layers. Residual connections help preserve low-level features while enabling the network to build deeper abstractions.

For fake face detection, this is particularly advantageous: **subtle anomalies** introduced by generative models may require deep hierarchical understanding, which ResNet can develop without suffering from vanishing gradients. It also allows generalization across face types and lighting conditions more effectively than shallow networks.[[11]](#footnote-11)

* **Strength:** Enables very deep models with high feature discrimination ability.
* **Benefit for our task:** Particularly effective in capturing nuanced details that reveal AI-generated inconsistencies.

### Inception Networks (GoogLeNet, Inception-v3)[[12]](#footnote-12)

Inception networks are designed for **multi-scale feature extraction**. Each Inception module applies multiple filters of different sizes (1×1, 3×3, 5×5) in parallel, allowing the network to analyze information at various spatial resolutions. The outputs are concatenated, giving a rich, diverse feature representation.

For real vs. fake face classification, this ability to process features at **multiple scales simultaneously** is useful. It can detect both global structure (head shape, eye alignment) and localized artifacts (noise near edges or eyes) that may be more prevalent in synthetic images.

* **Strength:** Multi-scale feature learning; efficient use of computational resources.
* **Challenge:** More complex to implement and tune compared to VGG or ResNet.

### MobileNet / EfficientNet[[13]](#footnote-13)

MobileNet and EfficientNet are lightweight CNN architectures optimized for performance on devices with limited processing power. MobileNet uses **depthwise separable convolutions**, which drastically reduce the number of parameters without a large drop in accuracy. EfficientNet introduces a **compound scaling** method, balancing depth, width, and resolution.

While these models are not specifically designed for detecting deepfake content, they provide a practical trade-off between **speed and accuracy**, making them useful for real-time or mobile deployment of a fake face detection system.

Suitability: Best for scaling or deploying the solution after evaluation with heavier models like ResNet or VGG.

* **Strength:** Compact, fast, and efficient.

### DenseNet (Densely Connected Convolutional Networks)[[14]](#footnote-14)

DenseNet builds upon ResNet by connecting each layer to **every other layer** in a feed-forward fashion. This ensures **maximum feature reuse**, strengthens gradient flow, and reduces the number of parameters compared to similarly deep networks.

For fake face detection, DenseNet's connectivity pattern allows it to retain and combine both **low-level textures and high-level representations**, which is useful when the features that differentiate real and fake are very subtle and dispersed across the image.

* **Advantage:** Strong feature propagation and efficient parameter use.
* **Benefit for our task:** Enhances model sensitivity to both low-contrast textures and unnatural facial patterns.

## Modern architectures comparison

Table 1: Modern architectures used to solve the problem.

|  |  |  |  |
| --- | --- | --- | --- |
| Architecture | Description and number/types of layers | Advantages | Disadvantages |
| **VGGNet (VGG16/VGG19)[[15]](#footnote-15)** | Deep CNN with 16 or 19 layers using only 3×3 convolutional filters, followed by max pooling and fully connected layers. | Easy to implement and modify; strong benchmark for image classification tasks. | Requires large memory and computational resources; lacks architectural efficiency. |
| **ResNet (Residual Network)[[16]](#footnote-16)** | Deep CNN ( ResNet-50, ResNet-101) with residual (skip) connections allowing for identity mapping and deeper training. | Enables training of very deep networks; mitigates vanishing gradient problem; strong feature learning capacity. | More complex architecture; increased training time and model tuning effort. |
| **Inception (GoogLeNet, Inception-v3)[[17]](#footnote-17)** | Modular network combining 1×1, 3×3, and 5×5 filters in parallel branches; often over 20 layers deep. | Multi-scale feature extraction; efficient use of parameters and computation. | Implementation complexity; harder to interpret and fine-tune compared to simpler models. |
| **MobileNet / EfficientNet[[18]](#footnote-18)** | Lightweight CNNs using depthwise separable convolutions (MobileNet) or compound scaling (EfficientNet); layers vary by variant. | Low latency and power consumption; well-suited for mobile and embedded applications. | May underperform on high-resolution or complex datasets; lower capacity than heavier architectures. |
| **DenseNet (Densely Connected CNN)[[19]](#footnote-19)** | CNN where each layer is connected to all subsequent layers (dense connectivity); typically 121–201 layers. | Improves gradient flow and feature reuse; achieves high accuracy with fewer parameters. | Increased memory usage due to concatenated feature maps; slower training for large models. |

# Models’ development and training

## Dataset

For this project, we utilized the “140k Real and Fake Faces” dataset, publicly available on Kaggle. The dataset is specifically designed for binary classification tasks and is perfectly suited for our objective of distinguishing between authentic and AI-generated human faces.

The dataset contains a total of 140,000 color facial images, evenly divided into:

* 70,000 real images, collected from Flickr’s high-quality facial datasets.
* 70,000 fake images, synthetically generated using StyleGAN—a state-of-the-art generative adversarial network capable of producing photo-realistic human faces.

Each image in the dataset is a centered face portrait, provided in JPEG format with a standardized resolution of 128×128 pixels, ensuring consistent input dimensions across all models. The images are also well-aligned, making them ready for use in CNN training without the need for complex face detection or cropping algorithms.

### **Dataset Preparation**

Before training the neural network models, several preprocessing steps were performed to prepare the dataset and ensure its suitability for image classification. First, the original dataset was significantly large, so we reduced it to a manageable size of 5,000 images, maintaining class balance by selecting 2,500 real and 2,500 fake images across training, validation, and test sets. Specifically, 2,500 training images (1,250 per class), 1,250 validation images (625 per class), and 1,250 test images (625 per class) were retained using a random sampling strategy.

Since the images were already standardized to 128×128 pixels in the original dataset, we maintained this resolution during preprocessing. However, we still explicitly included resizing in the transformation pipeline to ensure consistency and robustness in case of image distortion during augmentation. To enhance the model's generalization ability, several data augmentation techniques were applied to the training set - including random horizontal flips, slight rotations (up to 10 degrees), and controlled adjustments in brightness and contrast. These augmentations helped simulate natural variations in lighting and orientation commonly found in real-world data.

Finally, the images were converted into tensors and loaded into PyTorch (DataLoader) objects with a batch size of 32. This setup enables efficient batching, shuffling during training, and parallelized data loading, which is essential for scalable model training. Overall, this preprocessing pipeline helped ensure both computational efficiency and model reliability.

## Training and validation

### 3-way Splitting

The dataset used in this project was originally pre-split into three distinct subsets: training, validation, and testing. This three-way split is crucial for building, tuning, and evaluating a robust machine learning model. The training set is the amount of data the model sees during learning - it’s where the model adjusts its internal weights to minimize the loss function. The validation set, on the other hand, is used during training but not for updating the model. Instead, it serves as a checkpoint to monitor the model's performance on unseen data, helping prevent overfitting and guide hyperparameter tuning. Finally, the test set remains completely untouched during training. It is used only at the end of the process to evaluate how well the final trained model generalizes to truly unseen data.

In our case, we maintained a balanced class distribution across all three splits, and further reduced the size of each to create a manageable and efficient training workflow. Specifically, we used 2,500 images for training (50% of the reduced dataset), 1,250 for validation (25%), and 1,250 for testing (25%). This setup provided a reliable framework for measuring both the learning progress and the real-world predictive performance of each neural network architecture.

### hyperparameters for each architecture and their values

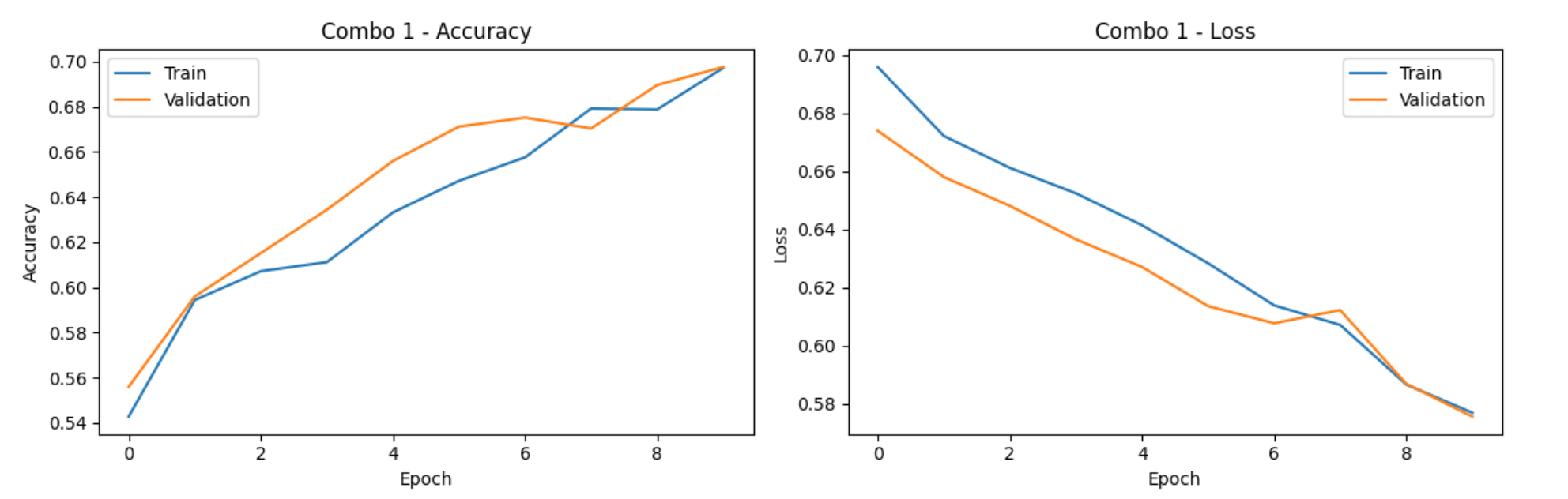
|  |  |  |  |
| --- | --- | --- | --- |
| **Architecture** | **Hyper-parameter** | **Description** | **Value(s)** |
| **SimpleNN** | **Learning rate** | controls how quickly the model updates its weights | 0.001, 0.0005, 0.01 |
| **optimizer** | defines the algorithm used to update the model weights. | Adam, SGD |
| **Drop out** | helps prevent overfitting by randomly deactivating neurons during training | 0.5, 0.3 |
| **VGGnet16** | **Learning rate** | controls how quickly the model updates its weights | 0.0001, 0.0005, 0.00001 |
| **optimizer** | defines the algorithm used to update the model weights. | Adam, SGD |
| **Drop out** | helps prevent overfitting by randomly deactivating neurons during training | 0.3, 0.4, 0.5 |
| **RESnet18** | **Learning rate** | controls how quickly the model updates its weights | 0.0001, 0.001, 0.0005 |
| **optimizer** | defines the algorithm used to update the model weights. | Adam, SGD |
| **DENSnet121** | **Learning rate** | controls how quickly the model updates its weights | 0.0001, 0.0005, 0.0003 |
| **optimizer** | defines the algorithm used to update the model weights. | Adam, SGD |

### Combinations of the hyperparameter values and corresponding performance achieved

|  |  |  |  |
| --- | --- | --- | --- |
| **Architecture** | **Combination of hyperparameter values** | **Training performance** | **Validation performance** |
| **SimpleNN** | {"lr": 0.001, "dropout": 0.5, "optimizer": "Adam"} | Train Acc: 0.6972 | Val Acc: 0.6976 |
| {"lr": 0.0005, "dropout": 0.3, "optimizer": "SGD"} | Train Acc: 0.5960 | Val Acc: 0.6136 |
| {"lr": 0.01, "dropout": 0.5, "optimizer": "Adam"} | Train Acc: 0.4996 | Val Acc: 0.5000 |
| **VGGnet16** | {"lr": 0.0001, "dropout": 0.3, "optimizer": "Adam"} | Train Acc: 0.8104 | Val Acc: 0.7576 |
| {"lr": 0.0005, "dropout": 0.4, "optimizer": "SGD"} | Train Acc: 0.7308 | Val Acc: 0.7160 |
| {"lr": 0.00001, "dropout": 0.5, "optimizer": "Adam"} | Train Acc: 0.7692 | Val Acc: 0.7472 |
| **RESnet18** | {"lr": 0.0001, "optimizer": "Adam"} | Train Acc: 0.6552 | Val Acc: 0.6608 |
| {"lr": 0.001, "optimizer": "SGD"} | 0.7160 | Val Acc: 0.7016 |
| {"lr": 0.0005, "optimizer": "Adam"} | Train Acc: 0.6992 | Val Acc: 0.6984 |
| **DenseNet121** | {"lr": 0.0001, "optimizer": "Adam"} | Train Acc: 0.6928 | Val Acc: 0.6992 |
| {"lr": 0.0005, "optimizer": "SGD"} | Train Acc: 0.7400 | Val Acc: 0.7344 |
| {"lr": 0.0003, "optimizer": "Adam"} | Train Acc: 0.7116, | Val Acc: 0.7288 |

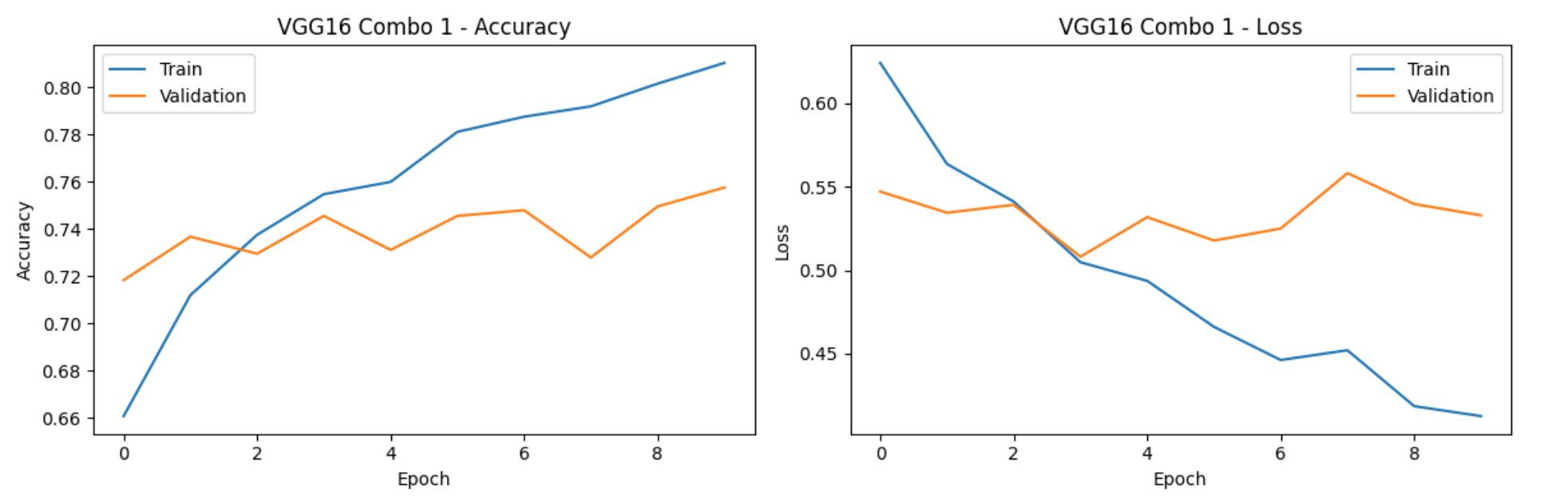
## learning curve (training and validation performance vs epochs for each model (best)

### SimpleNN



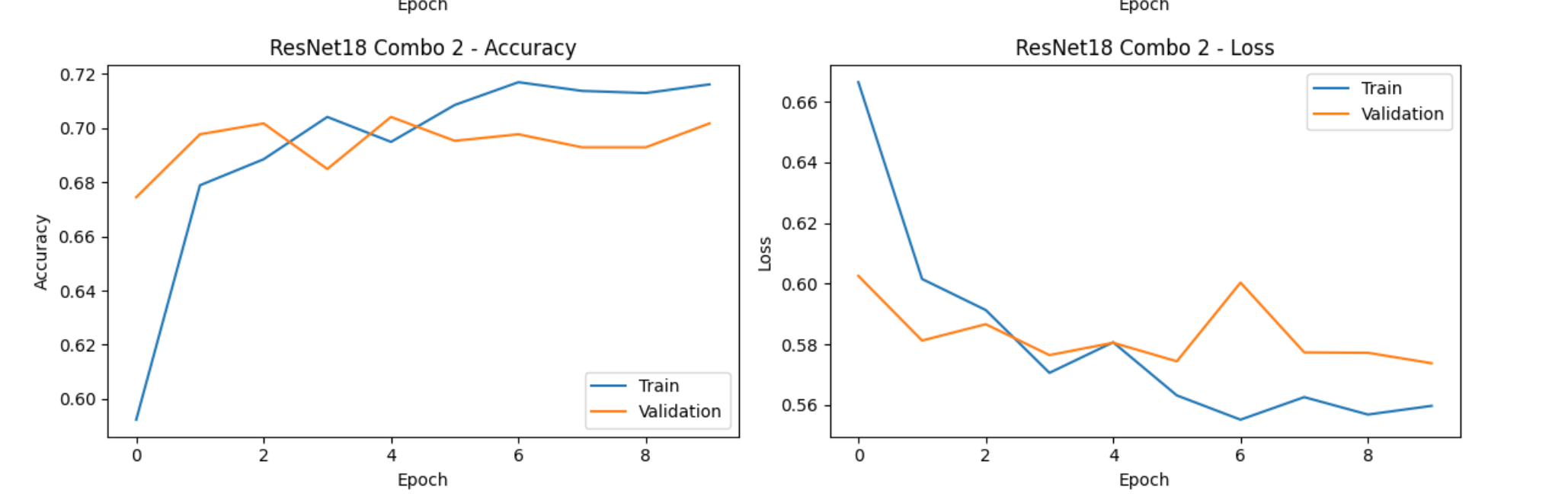
The two charts show how well the SimpleCNN model learned using Combo 1. In the first chart, the accuracy improves over time for both training and validation, reaching about 70% by the end. In the second chart, the loss gets lower, which means the model is making fewer mistakes. Since both training and validation follow a similar pattern, the model is learning well and not overfitting.

### GVVnet16



The charts show how the VGG16 model performed using Combo 1. The accuracy chart on the left shows that the training accuracy improved steadily, while the validation accuracy stayed mostly stable with some small ups and downs. The loss chart on the right shows that the training loss decreased over time, but the validation loss didn’t drop as much and was less consistent. This suggests the model is learning well on the training data, but may not be improving as much on new, unseen data.

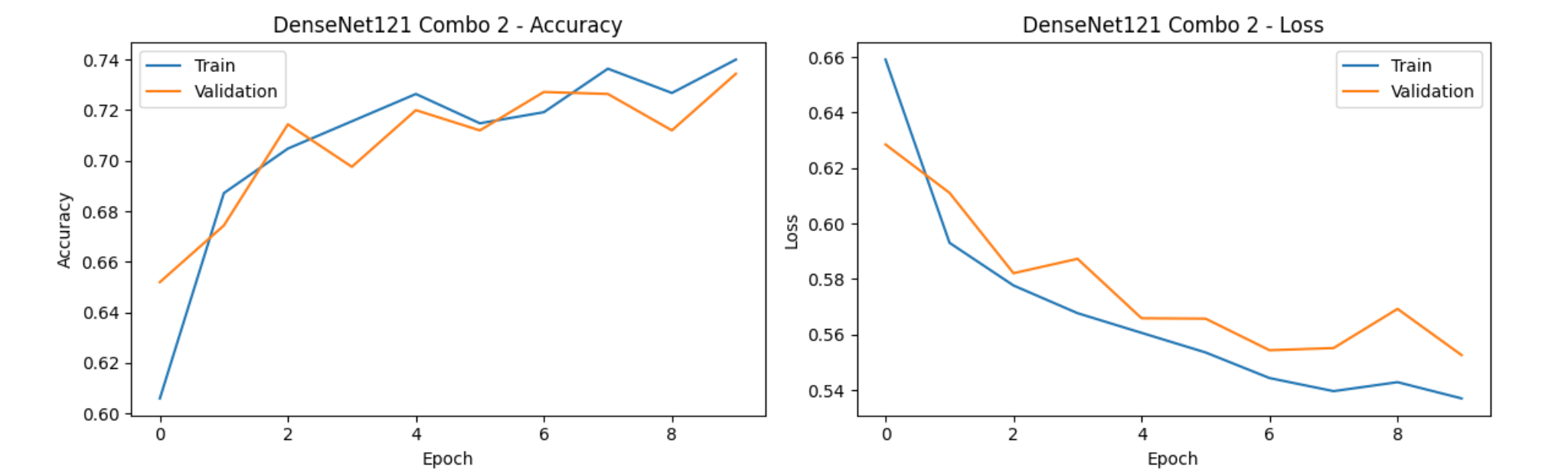
### RESnet18



On the left, we see that training accuracy improved steadily and reached about 71.8%, while validation accuracy remained slightly lower and more stable, around 70%. This means the model learned patterns from the training data, but didn’t overfit much - which is good.

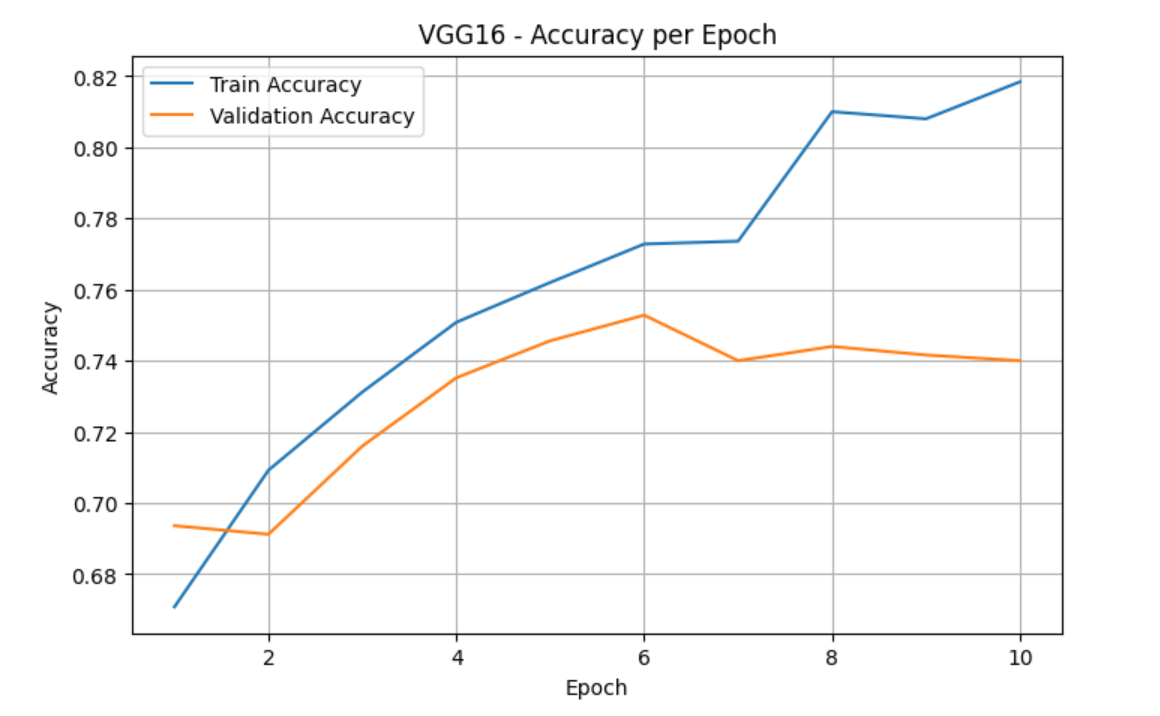
The right chart shows that the training loss dropped clearly over time, while validation loss decreased but stayed a bit noisy. This suggests the model learned effectively, and though there may still be room for tuning, it generalizes well to new data.

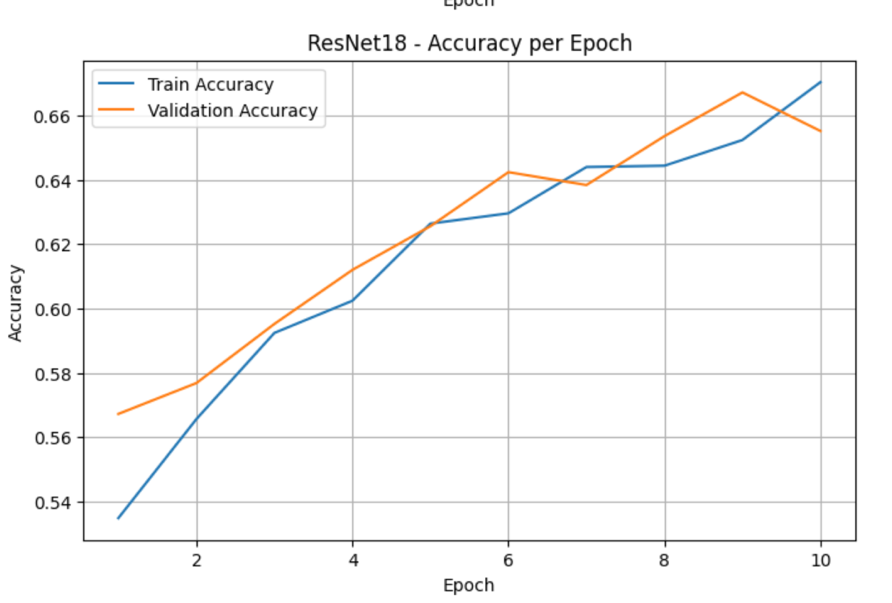
### DenseNet121

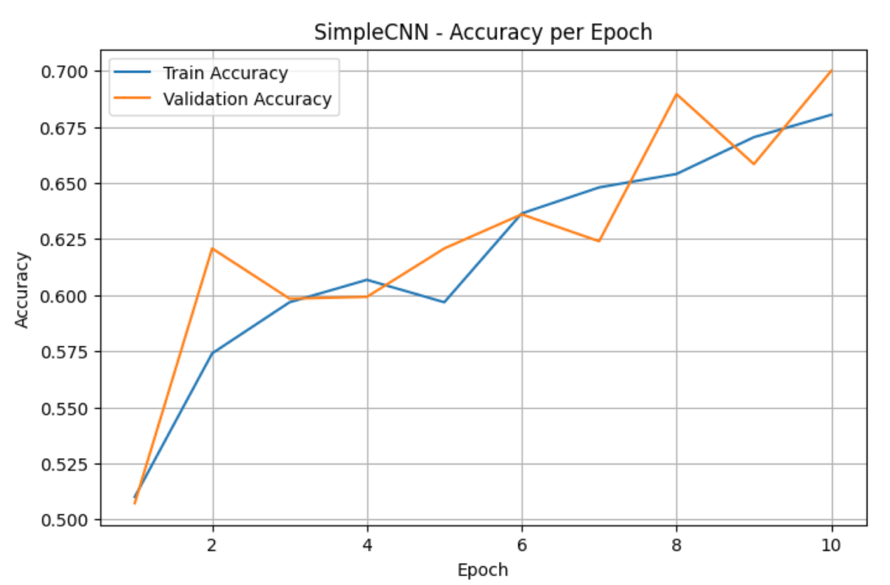


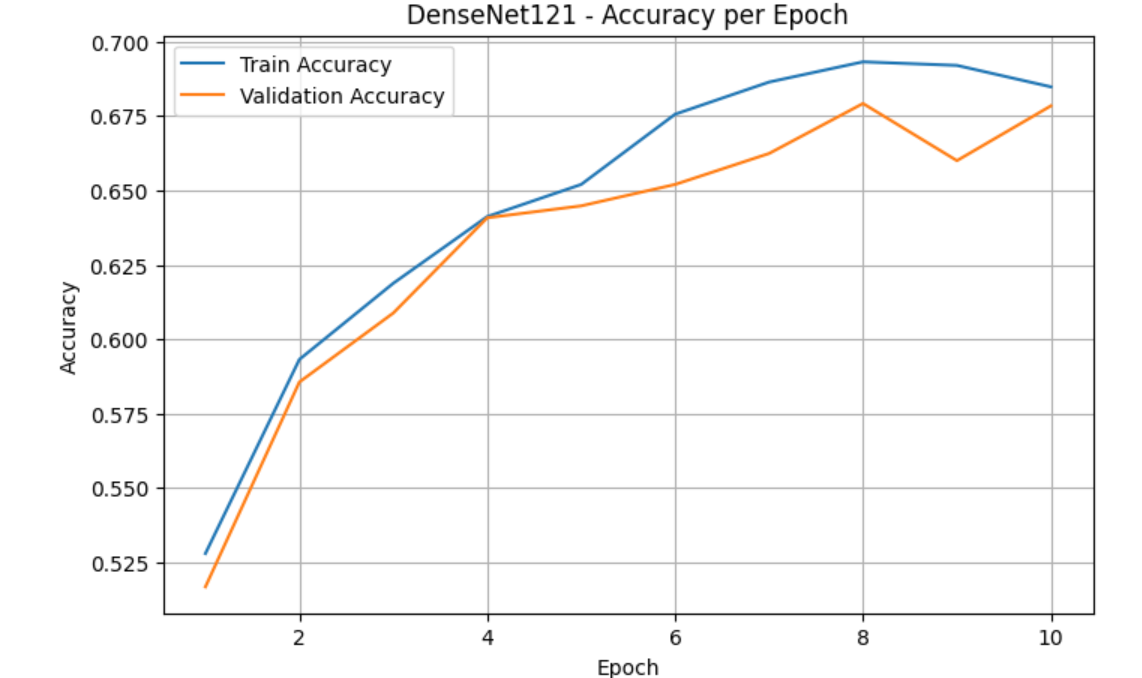
The graphs show how the DenseNet121 model performed with Combo 2. In the accuracy chart, both the training and validation accuracy improved steadily, reaching around 74%. In the loss chart, both training and validation loss decreased over time, with training loss going slightly lower. The curves follow a similar trend, which means the model is learning well and generalizing nicely without overfitting.

## Epochs vr Acurracy for the best combo of the 4 models

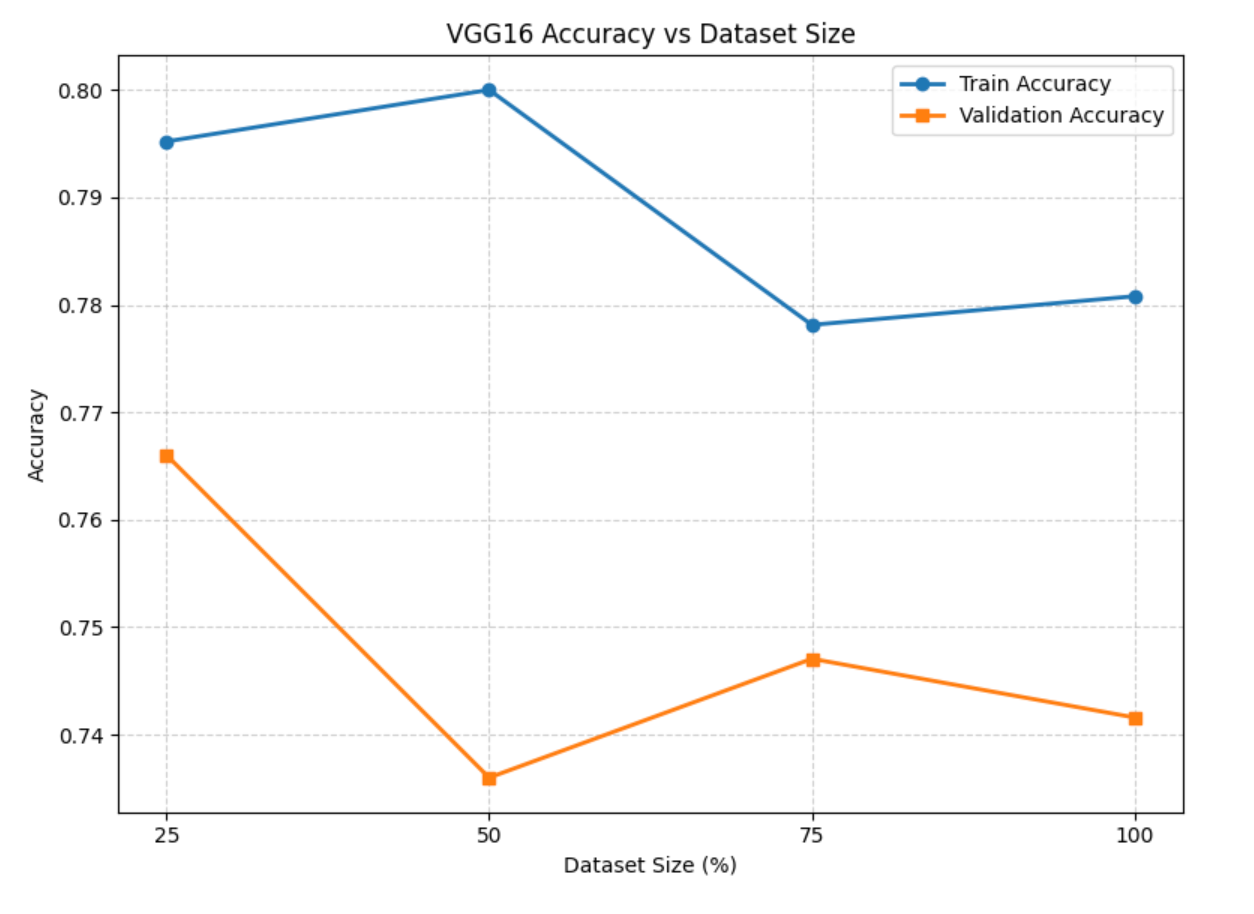








### Datasize vs Acurracy for best model



# Models’ testing and evaluation

## Testing

After training all four models—**SimpleCNN, VGG16, ResNet18, and DenseNet121**—the best-performing version (best hyperparameter combo) of each architecture was evaluated using the **test dataset**. This dataset was kept completely unseen during training and validation to ensure an honest and unbiased assessment of each model’s generalization ability.

For each model, we reloaded the best version (saved earlier) and passed the test images through it. Predictions were generated using the trained model, and these outputs were then compared with the true labels from the test set. The threshold for binary classification was set to 0.5, meaning any predicted probability above 0.5 was classified as "real," and below 0.5 as "fake."

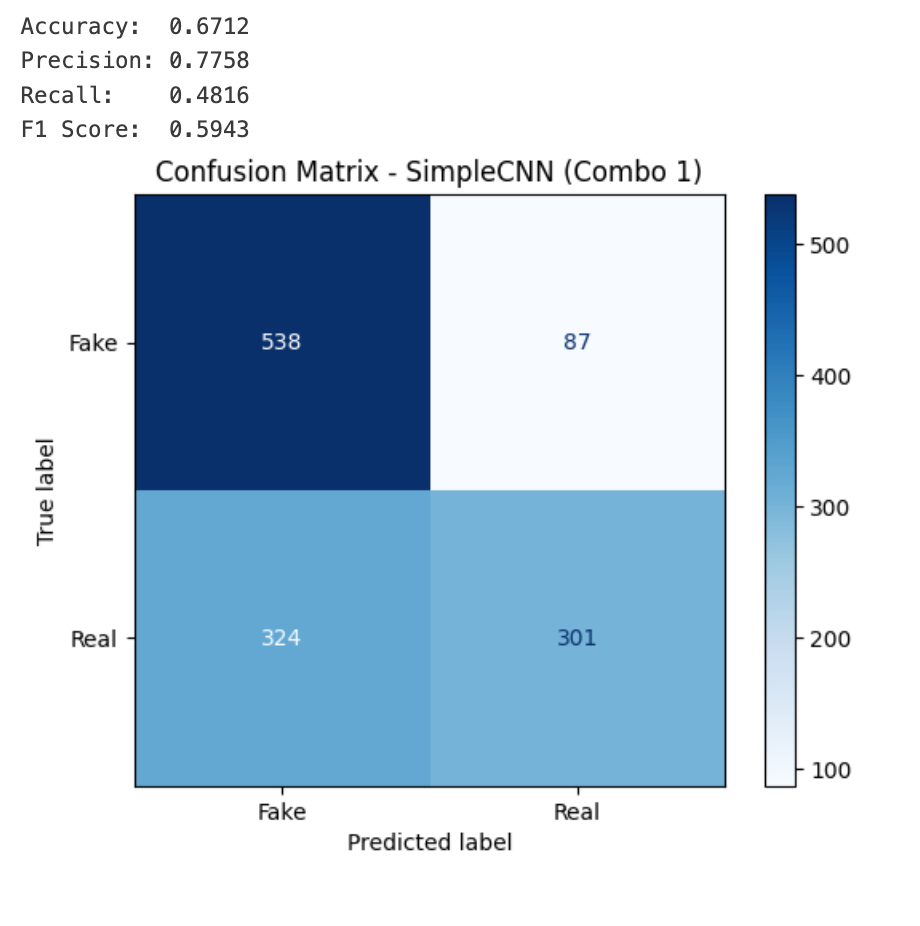
## **Evaluation Metrics Chosen**

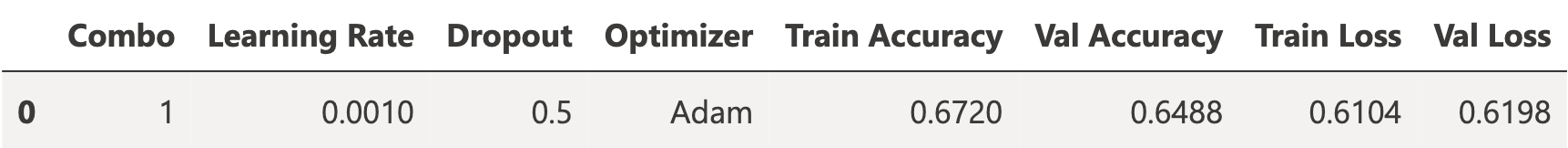
To measure how well each model performed, we used the following standard metrics:

* **Accuracy**: This shows the overall correctness of the model. It's the percentage of total predictions that were right. It's helpful for getting a quick sense of performance but doesn’t tell the full story in cases of class imbalance.
* **Precision**: This tells us how many of the predicted "fake" faces were actually fake. High precision means the model is cautious and avoids falsely labeling real faces as fake.
* **Recall**: Also known as sensitivity, this tells us how many of the actual fake faces the model managed to detect. A high recall indicates that the model is good at catching fakes, even if it makes more mistakes.
* **F1 Score**: This combines precision and recall into a single number, giving a balanced view. It’s especially useful when we care equally about avoiding both false positives and false negatives.
* **Confusion Matrix**: We also plotted the confusion matrix to get a visual summary of how the model performed on each class. It shows the count of true positives, true negatives, false positives, and false negatives, helping to understand exactly where the model got confused.

## Values for the evaluation metrics achieved on the test set for the best model from each architecture.

### SimpleNN



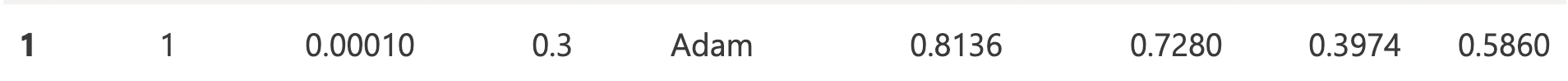


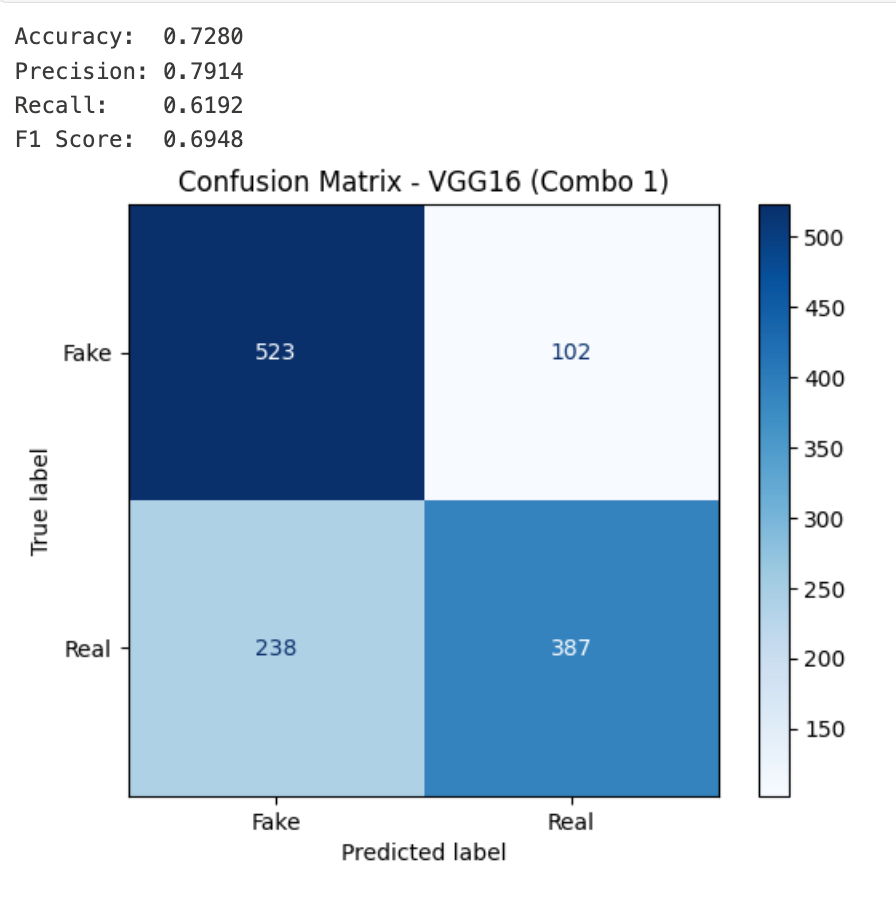
The SimpleCNN model (Combo 1) was evaluated on the test set, and its performance showed a mix of strengths and limitations. The model achieved an overall accuracy of **67.12%**, which means it correctly classified about two-thirds of the images. One of its strong points is **precision**, which reached **77.58%**—this tells us that when the model predicts an image is fake, it’s usually correct. In other words, it’s good at avoiding false positives.

However, the model struggled with **recall**, scoring just **48.16%**. This means it missed more than half of the fake images, misclassifying them as real. The **F1 score**, which balances precision and recall, came out to **59.43%**, reflecting the model’s overall moderate ability to correctly detect fake faces without making too many mistakes.

Looking at the confusion matrix, the model correctly identified **538 fake images** and **301 real ones**. However, it also misclassified **87 fake images as real**, and **324 real images as fake**. This shows the model is cautious—it tends to err on the side of calling images fake, which explains the high precision but lower recall.

### VGGnet16



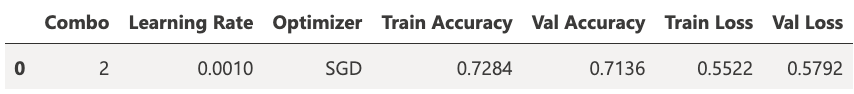


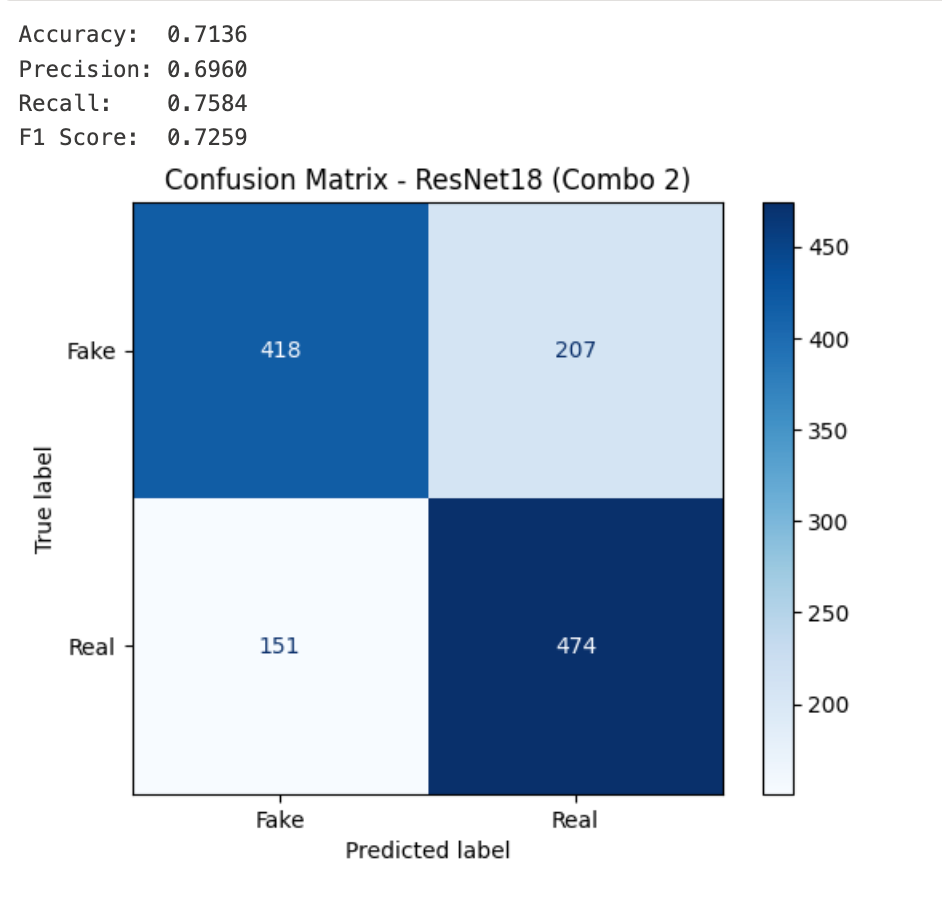
The VGG16 model using Combo 1 showed a solid performance during testing. It reached an accuracy of **72.80%**, which means it correctly classified almost 3 out of every 4 images. The model also had a **precision of 79.14%**, indicating that when it predicted a face as fake, it was right nearly 80% of the time—this shows a good ability to avoid false positives.

In terms of **recall**, the model scored **61.92%**, which is better than the SimpleCNN’s performance. This means it was able to catch more fake images than before, though it still missed a good portion. The **F1 score**, which balances precision and recall, landed at **69.48%**, showing that the model maintains a strong trade-off between catching fake faces and not misclassifying real ones.

Looking at the **confusion matrix**, we see that the model correctly identified **523 fake** images and **387 real** ones. However, it mistakenly predicted **102 fake** faces as real, and **238 real** faces as fake. Compared to the SimpleCNN, this model shows a better balance between the two classes. It's not overly cautious nor too lenient—it manages to improve recall without sacrificing too much precision, making it more reliable overall.

### RESnet18





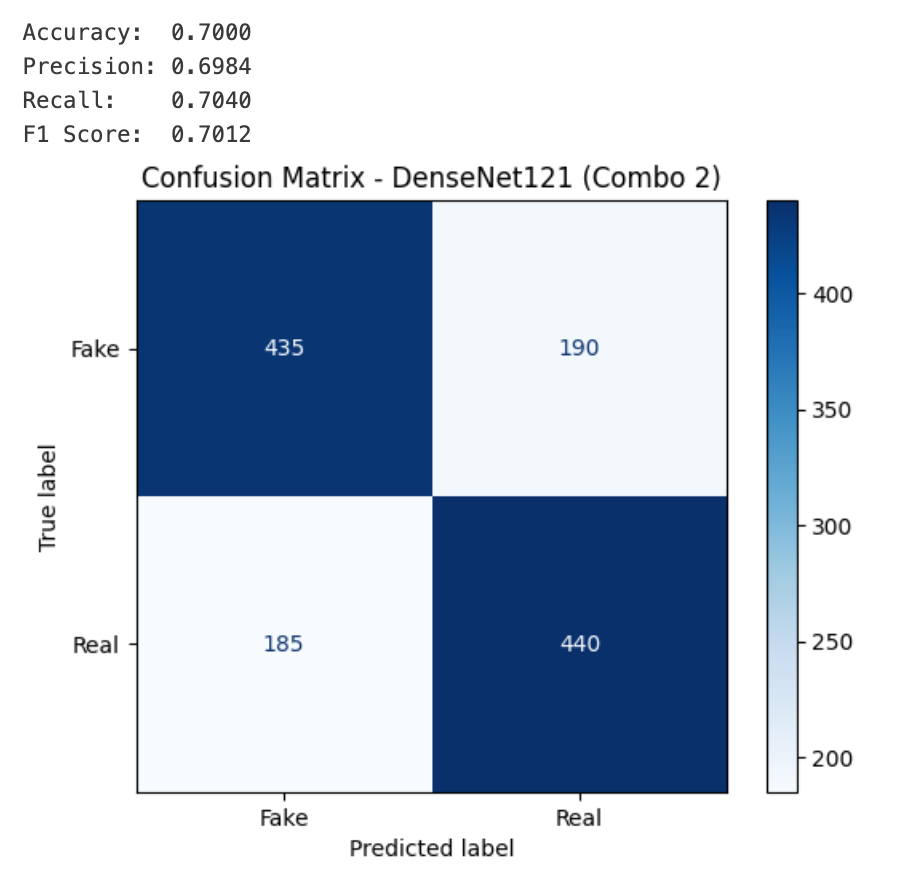
The ResNet18 model using Combo 2 demonstrated strong and consistent performance. It achieved a test accuracy of **71.36%**, which means it correctly classified over 7 out of every 10 images. Its **precision** came in at **69.60%**, showing that most of its predictions for fake images were correct. Even more impressively, the model reached a **recall** of **75.84%**, meaning it successfully detected a large portion of the actual fake images—better than any model before it. Its **F1 score** was **72.59%**, indicating a strong balance between precision and recall.

In the **confusion matrix**, we see that the model correctly predicted **418 fake** and **474 real** images. It misclassified **207 fake** images as real, and **151 real** images as fake. Compared to SimpleCNN and VGG16, ResNet18 does a much better job at catching fakes while still being fairly accurate at identifying real ones.

Overall, this model offers a strong combination of generalization and reliability. It doesn't overcommit to one class and has a balanced ability to detect both real and AI-generated faces, making it one of the most dependable models in this experiment.

### denseNet121





The DenseNet121 model with Combo 2 performed well on the test set, reaching a solid **accuracy of 70.00%**. This indicates that 7 out of every 10 predictions made by the model were correct. Its **precision** was **69.84%**, meaning that when the model predicted a face as fake, it was right nearly 70% of the time. Similarly, its **recall** was **70.40%**, showing that it successfully identified about 70% of the actual fake faces. These balanced values led to an **F1 score of 70.12%**, reflecting consistent performance across both precision and recall.

Looking at the **confusion matrix**, we see that the model correctly predicted **435 fake** and **440 real** images. It misclassified **190 fake** images as real and **185 real** images as fake. This shows a fairly even and balanced ability to identify both real and fake faces, without strongly favoring one class over the other.

Overall, DenseNet121 demonstrates reliable and balanced performance, making it a strong choice for binary classification tasks where both classes are equally important to detect.

## Over/under-fitting assessment

### SimpleNN

Looking at **Combo 1**, which had the best performance, the training accuracy increased steadily from 55.68% in Epoch 1 to 67.20% in Epoch 10. The validation accuracy followed a similar pattern, starting at 58.48% and peaking around 65.52%, though it slightly dropped by the final epoch. The closeness between the training and validation accuracy curves suggests the model is not significantly overfitting—both sets show consistent improvement without a large gap. This is further supported by the test performance: an accuracy of **67.12%**, precision of **77.58%**, recall of **48.16%**, and an F1-score of **59.43%**. These results show that the model performs reasonably well on unseen data, although it still struggles a bit with correctly identifying real faces (lower recall).

On the other hand, **Combo 2** and **Combo 3** demonstrate signs of **underfitting**. Combo 2 shows low training and validation accuracy across epochs (e.g., training ended at 58.52%, validation at 57.52%), indicating the model wasn't able to learn enough patterns from the data. Combo 3, in particular, is flatlined around 50% accuracy for both training and validation across all epochs, which is essentially random guessing in a binary classification task—suggesting either the learning rate was too high or the model was not able to converge at all.

In conclusion, **SimpleCNN Combo 1** achieved a fairly balanced learning outcome with modest accuracy and minimal overfitting, making it the most reliable configuration among the three.

### VGGnet16

Among the three configurations tested, **Combo 2** (Learning rate = 0.0005, Dropout = 0.4, Optimizer = SGD) produced the best overall performance, particularly on the validation set where it achieved an accuracy of **74.08%**, closely matching the training accuracy of **73.92%**. This closeness between training and validation performance is a strong sign that the model is **not overfitting** and has achieved a good balance in learning.

Looking at **Combo 1**, while the training accuracy climbed to **81.36%**, the validation accuracy peaked lower at **72.80%**, creating a noticeable gap. This gap, along with a low validation F1-score of **0.6948** and a precision of **0.7914**, indicates **mild overfitting**—the model learned the training data well but started to lose generalization capability. However, it's still a strong performer on the test set, with a **test accuracy of 72.80%**.

**Combo 3**, on the other hand, showed consistent training and validation accuracy (around **77.20%** and **72.48%**, respectively), and even though its performance was slightly lower than Combo 2, it still demonstrates good generalization and no signs of underfitting.

In summary, **VGG16 Combo 2** provides the most balanced results, showing no overfitting and a steady learning pattern. **Combo 1** is still a strong model, especially in terms of precision, but the slight overfitting could affect its reliability on completely unseen data.

### RESnet18

In evaluating ResNet18, **Combo 2** (Learning Rate: 0.001, Optimizer: SGD) stands out as the best-performing configuration. The training accuracy reached **72.84%**, while the validation accuracy was close at **71.36%**, with similarly aligned loss values. This small gap between training and validation performance suggests that **the model is not overfitting**—it generalizes well and is not simply memorizing the training data. The stability across training and validation curves indicates that Combo 2 strikes a good balance between learning enough patterns and avoiding noise.

On the **test set**, Combo 2 achieved **71.36% accuracy**, **69.60% precision**, **75.84% recall**, and an **F1-score of 72.59%**. These results confirm that the model performs reliably on unseen data. The confusion matrix further shows balanced classification performance, with strong detection of real and fake faces. The high recall suggests that the model is effective in catching most fake faces, while its precision indicates it makes relatively few false positive predictions.

By contrast, **Combo 1** shows **signs of underfitting**. It starts with low training and validation accuracy (below 50%) and ends with around **66% training** and **65% validation accuracy**. The consistent gap between where the model should be and where it ends up suggests it struggles to capture enough features from the data, possibly due to a learning rate that’s too low or an insufficient number of trainable layers.

**Combo 3** also performs well, with **71.52% training accuracy** and **70.72% validation accuracy**, and shows no major signs of overfitting. However, its performance is slightly less stable than Combo 2, especially across some epochs where the validation curve fluctuates more.

### denseNet121

For DenseNet121, **Combo 2** (Learning Rate: 0.0005, Optimizer: SGD) showed the best performance among the three configurations. The model’s **training accuracy reached 74.68%**, with **validation accuracy closely following at 71.52%**, and both loss values remained quite close (Train Loss: 0.5280, Val Loss: 0.5603). The narrow gap between these values suggests that the model **generalizes well and does not suffer from overfitting**. It manages to learn patterns effectively without simply memorizing the training data.

When evaluated on the test set, the model achieved **70.00% accuracy**, with **precision at 69.84%**, **recall at 70.40%**, and an **F1-score of 70.12%**. These results show balanced performance across all evaluation metrics, meaning the model can identify both real and fake faces with similar reliability. The confusion matrix supports this, with 435 fake faces and 440 real faces correctly classified, while the number of misclassifications (190 and 185) remains reasonably low and balanced.

Comparatively, **Combo 1** (with a lower learning rate) ended with training and validation accuracy around 69–70%, but it showed slower learning progress during training and slightly higher loss, indicating that it may be **slightly underfitting**—not fully optimizing its performance during the learning process.

**Combo 3** had strong final performance as well (Train Accuracy: 71.68%, Val Accuracy: 72.24%), and even slightly higher validation accuracy than Combo 2, but its training process was less consistent across epochs, with more fluctuations. This could suggest **mild instability** in learning, possibly due to the Adam optimizer or the smaller learning rate.

DenseNet121 Combo 2 shows no signs of overfitting or underfitting. It maintains a healthy alignment between training and validation performance, and its test results confirm that the model can generalize effectively to unseen data.

## Results analysis

After training and evaluating all four models—SimpleCNN, VGG16, ResNet18, and DenseNet121—it’s clear that the **deeper and more modern architectures** consistently outperformed the simpler baseline.

SimpleCNN, while lightweight and fast to train, showed **limited capacity to capture complex patterns** in the data. It served well as a benchmark, but struggled with generalizing to more subtle features that distinguish real from fake faces.

VGG16 and ResNet18, on the other hand, demonstrated **stronger performance across metrics**, with ResNet18 showing the **best overall balance** between detecting real and fake images. Its use of residual connections helped it learn effectively without overfitting.

DenseNet121 also performed well, offering **stable learning behavior and solid generalization**, though it didn’t surpass ResNet18. It benefited from feature reuse and efficient parameter handling.

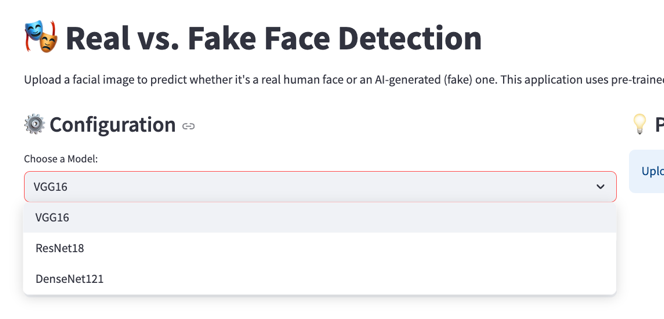
Overall, the experiment shows that **deeper CNNs with advanced design patterns like skip or dense connections are better suited for detecting fake faces**, as they extract richer features and handle complexity more effectively. Simpler models can be useful for quick experimentation, but they tend to underperform in more challenging classification tasks like this one.

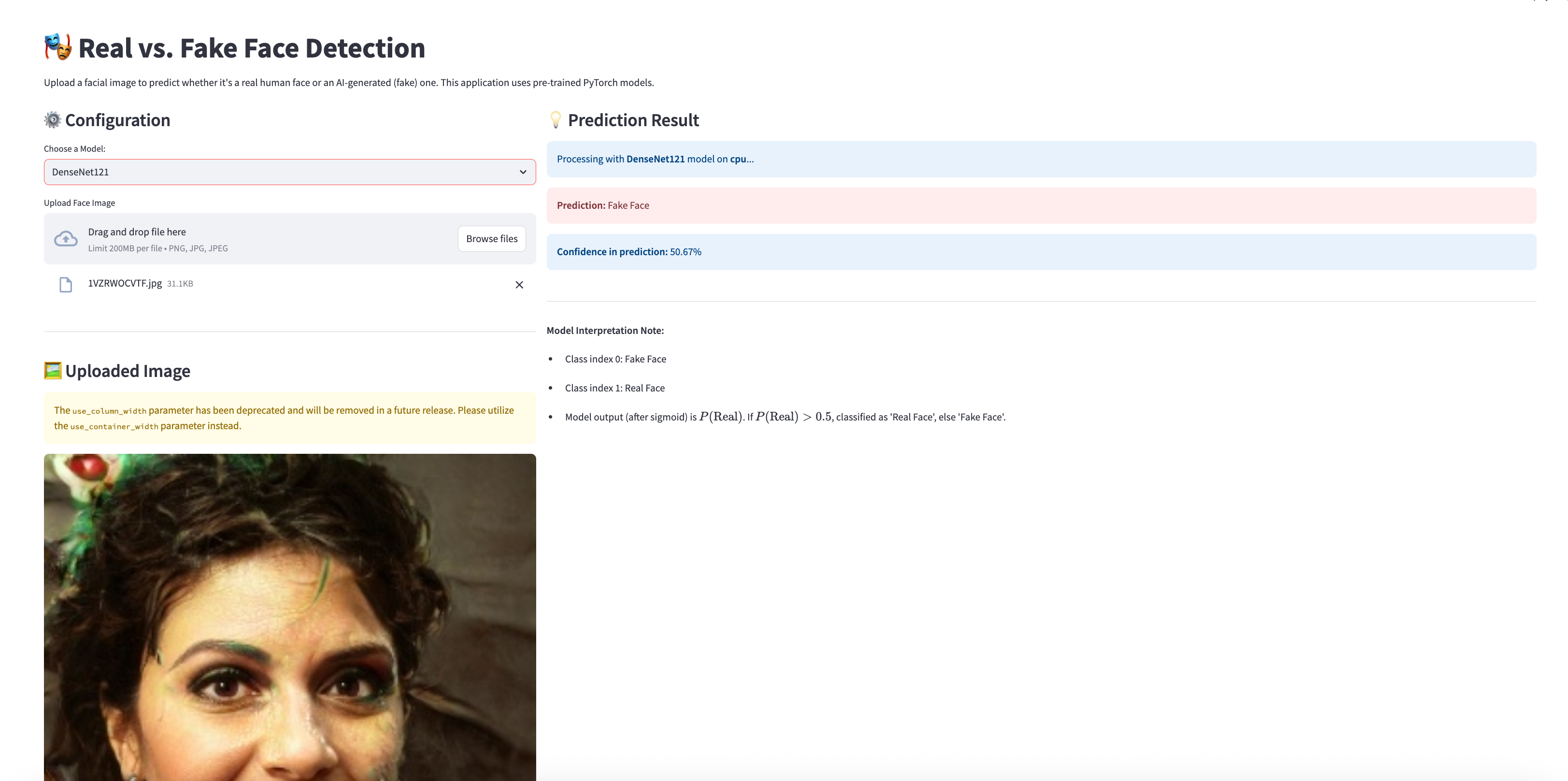
## Effectiveness assessment

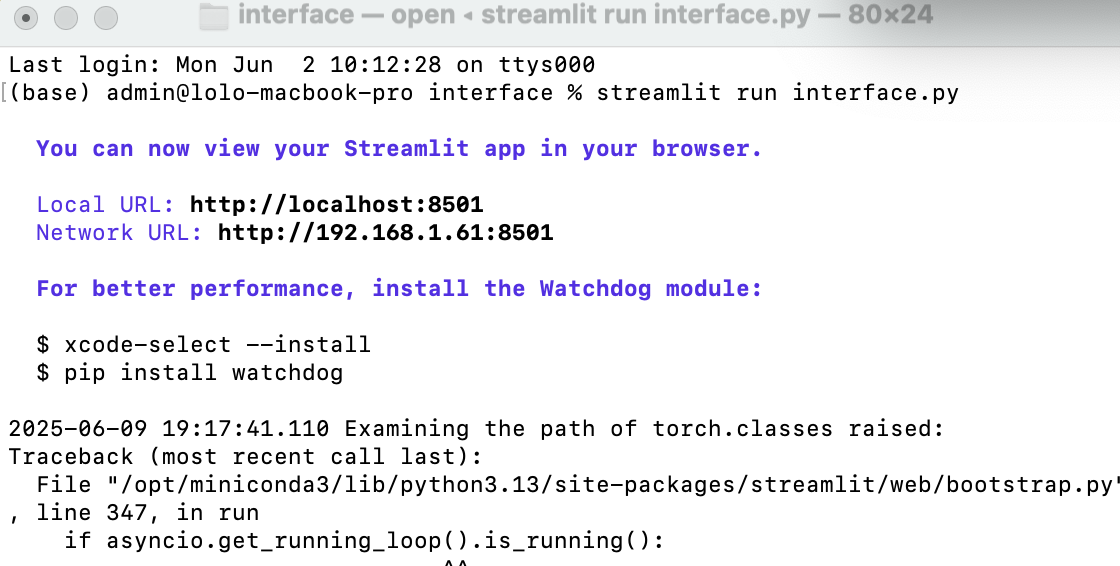
* **SimpleCNN** was the most lightweight and fastest to train. Its architecture has very few layers and parameters, making it highly efficient in terms of both memory and training speed. This simplicity comes at the cost of performance, especially on more complex visual tasks.
* **VGG16**, although relatively old compared to modern architectures, has **a very large number of parameters** due to its fully connected layers. This leads to **high memory consumption** and slower training times. Despite that, it produced decent performance, especially in capturing fine-grained features.
* **ResNet18** offered an **excellent balance** between performance and efficiency. It has significantly fewer parameters than VGG16 but benefits from its residual (skip) connections that allow deeper training. It trained relatively fast and had moderate memory requirements, making it a strong candidate for practical applications.
* **DenseNet121**, while deep and accurate, is **computationally heavier** due to its dense connectivity pattern. It reuses features extensively, which improves accuracy and gradient flow but also increases training complexity and memory usage. It is effective but more suited for environments with stronger compute power.

# Interface development

We can choose the model from here:



And then upload the picture and it will make prediction 



The interface was successfully developed and launched using Streamlit, as shown in the terminal output. The application is accessible locally via <http://localhost:8501> and on the local network via http://192.168.1.61:8501.  
This confirms that the script interface.py is running, and the Streamlit app has been deployed.

# Critical evaluation of models

This project implemented and evaluated four deep learning models, SimpleCNN, VGG16, ResNet18, and DenseNet121, for the task of detecting AI-generated (fake) faces. A user-friendly Streamlit interface was also deployed, allowing real-time prediction from any of the saved models based on user-uploaded facial images.

**Best Model – VGG16:**  
Based on testing results, **VGG16 emerged as the best-performing model**, achieving the **highest test accuracy (72.80%)** and **highest precision (79.14%)**. This means it is most reliable at correctly identifying fake faces **without mistakenly labeling real faces as fake**. Its F1 score of 69.48% reflects a strong balance between precision and recall, making it an ideal choice for real-world use, where both **accuracy** and **trustworthiness** are critical.

**Real-Time Interface Deployment:**  
All four models were successfully deployed in a real-time Streamlit application. The interface allows users to:

* Upload a face image
* Choose one of the four trained models (from saved checkpoints)
* Receive a prediction (real or fake) immediately

**Usability and Efficiency Comparison:**

* **SimpleCNN**: Fastest and lightest, suitable for quick demo/testing. However, it underperformed in accuracy and recall.
* **VGG16**: Slightly slower to run and heavier on memory, but **delivers the best accuracy and precision**, making it the best choice for reliability.
* **ResNet18**: Offers high recall (75.84%), it catches more fakes, but slightly lower precision and accuracy than VGG16.
* **DenseNet121**: Balanced and stable, but heavier than ResNet18 and slightly behind VGG16 in all metrics

**Strengths of the Current Deployment:**

* Interface is easy to use and works in real time.
* Users can switch models and directly observe performance differences.
* The model’s predictions are based on fully trained versions stored in memory, ensuring consistency.

**Suggested Future Improvements:**

* Add **EfficientNet or MobileNetV3** for faster real-time predictions on mobile devices.
* Enable **automatic model selection** based on highest accuracy.
* Explore **auto-selection of the best model** based on device capacity or use-case requirements.

**Generalization and Robustness:**  
Evaluation metrics and confusion matrices indicate that **VGG16 generalizes best to unseen data**, achieving the highest overall test accuracy (72.80%) and precision (79.14%). These results demonstrate that VGG16 effectively captures and distinguishes the subtle differences between real and AI-generated faces, even on new inputs. Its balanced performance (F1 score: 69.48%) shows strong generalization, with minimal overfitting.

While **ResNet18** achieved the highest recall (75.84%)—meaning it catches more fake faces—it did so with slightly lower precision and overall accuracy compared to VGG16. **DenseNet121** also showed consistent and balanced generalization but didn’t outperform VGG16 in any specific metric.

The deployed **Streamlit interface** allows users to upload new images and instantly test all models, providing real-time confirmation that **VGG16 remains the most reliable and accurate model** in practical, unseen scenarios.

1. <https://en.wikipedia.org/wiki/Deepfake> [↑](#footnote-ref-1)
2. <https://arxiv.org/abs/1812.04948> [↑](#footnote-ref-2)
3. <https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks> [↑](#footnote-ref-3)
4. <https://www.geeksforgeeks.org/kernels-filters-in-convolutional-neural-network/> [↑](#footnote-ref-4)
5. <https://medium.com/@khwabkalra1/convolutional-neural-networks-for-image-classification-f0754f7b94aa> [↑](#footnote-ref-5)
6. <https://www.geeksforgeeks.org/stylegan-style-generative-adversarial-networks/> [↑](#footnote-ref-6)
7. <https://elearning.htu.edu.jo/pluginfile.php/157683/mod_resource/content/2/L06-%20Multilayer%20Perceptrons_2.pdf> [↑](#footnote-ref-7)
8. <https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4> [↑](#footnote-ref-8)
9. <https://pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/> [↑](#footnote-ref-9)
10. <https://www.geeksforgeeks.org/image-classification-using-resnet/> [↑](#footnote-ref-10)
11. <https://www.tandfonline.com/doi/full/10.1080/1206212X.2025.2465727> [↑](#footnote-ref-11)
12. <https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/> [↑](#footnote-ref-12)
13. <https://mljourney.com/resnet-vs-mobilenet-vs-efficientnet-dive-into-cnn-architectures/> [↑](#footnote-ref-13)
14. <https://arxiv.org/abs/1608.06993> [↑](#footnote-ref-14)
15. <https://arxiv.org/abs/1409.1556> [↑](#footnote-ref-15)
16. <https://arxiv.org/abs/1512.03385> [↑](#footnote-ref-16)
17. <https://arxiv.org/abs/1409.4842> [↑](#footnote-ref-17)
18. <https://arxiv.org/abs/1704.04861> [↑](#footnote-ref-18)
19. <https://arxiv.org/abs/1608.06993> [↑](#footnote-ref-19)