Projeto Bimestral

Erick Sartori 13381156

João Pedro Viana Dubiela 13262572

Vinicius Krieger Granemann 13416847

I. Introdução

Nesse relatório estudaremos as diferenças entre 3 CNN's diferentes aplicadas a mesma base de testes, a base de teste consiste na base *"Bird Species Classification 200 Categories"*, a base consiste em 200 espécies de pássaros separadas por pastas,cada uma contendo 5 imagens de cada pássaro, o que significa que a base está balanceada. Para a primeira e segunda CNN's nós utilizamos as arquiteturas ResNet e DenseNet, espectivamente. Fizemos as duas de 3 maneiras, sem transfer learning, com transfer learnig da forma que a primeira camada é congelada e utilizada no próximo treinamento e a ultima sendo o transfer learning congelar a última camada e utilizá-la no próximo treinamento e por fim criamos uma CNN com a nossa arquitetura. Os parâmetros foram definidos na nossa arquitetura para poder comparar com as arquiteturas já estabelecidas.

II. CNN customizada

Nossa CNN é definida com três camadas convolucionais seguidas por camadas de pooling, uma camada densa intermediária e uma camada de saída softmax.

III. Arquiteturas Prontas

Neste projeto nós utilizamos a ResNet e DenseNet como arquiteturas já consolidadas para comparação com a nossa CNN, A ResNet é uma arquitetura de rede neural convolucional (CNN) que foi introduzida para abordar o problema de degradação do desempenho à medida que a rede se torna mais profunda. Normalmente, espera-se que o desempenho de uma rede melhore à medida que sua profundidade aumenta. No entanto, nas CNNs convencionais, o aumento da profundidade leva a uma diminuição na precisão do treinamento devido ao desaparecimento do gradiente.
A ResNet resolve esse problema usando conexões residuais. Em vez de simplesmente empilhar camadas uma após a outra, a ResNet introduz caminhos de salto (skip connections) que permitem que a informação seja transmitida diretamente para camadas posteriores. Essas conexões residuais permitem que o gradiente se propague mais facilmente durante o treinamento, o que facilita o treinamento de redes mais profundas. A arquitetura básica da ResNet é composta por blocos residuais que contêm camadas convolucionais, seguidas por conexões de salto, e A DenseNet é uma arquitetura de CNN que se destaca por sua conectividade densa. Nessa arquitetura, todas as camadas estão conectadas umas às outras em um padrão denso, em vez de seguir a estrutura convencional de conexões sequenciais. Cada camada recebe entradas não apenas das camadas anteriores, mas também de todas as camadas subsequentes na rede.

IV. Imagens

As imagens utilizadas nesse projeto foram 200 espécies de pássaros dividas em pastas, cada imagem possui 24x24px, as imagens foram dividas em 70% para teste e 30% para validação, as imagens passaram por aumentador de dados, que ajudou a diversificar as imagens para teste,

```
train_gen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.3
)
```

,
esse fragmento de código contem a maneira que os dados foram aumentados,
rescale=1. / 255: Normaliza os valores dos pixels das imagens, dividindo todos os valores por 255, o que coloca os valores na faixa de 0 a 1.
shear_range=0.2: Aplica deformações de cisalhamento nas imagens com um fator máximo de 0.2. Isso pode inclinar ou distorcer as imagens.
zoom_range=0.2: Aplica zoom in e zoom out nas imagens com um fator máximo de 0.2. Isso pode aumentar ou reduzir o tamanho aparente das imagens.
horizontal_flip=True: Inverte as imagens horizontalmente, o que cria imagens espelhadas.

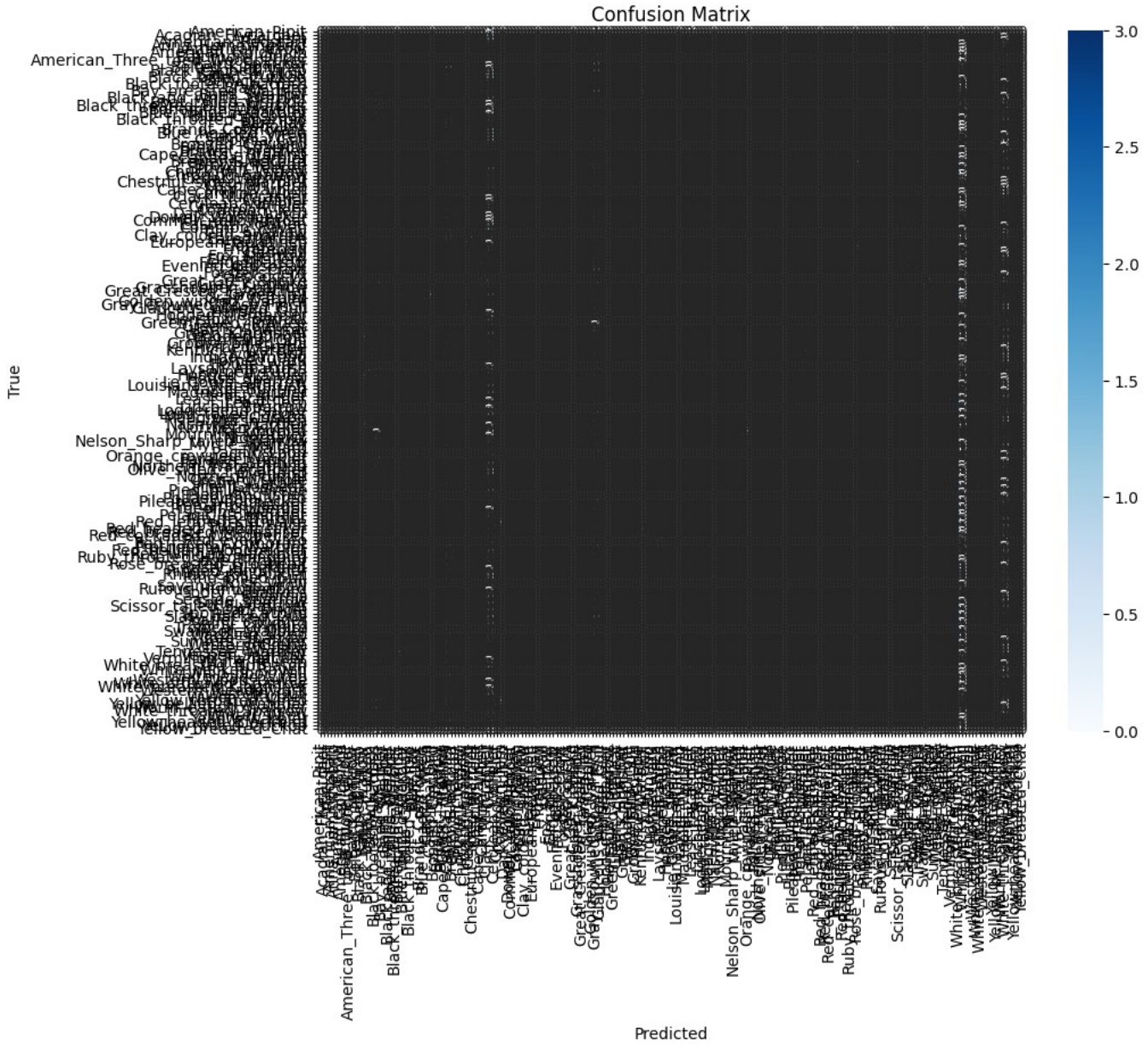
V. MODELOS

Para as arquiteturas de resnet e densenet essas foram as camadas de treino definidas mediante a testes feitos na com a resnet.

```
inputs = tf.keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
x = base_model(inputs, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)

#x = tf.keras.layers.Dense(240, activation='relu')(x)
#x = tf.keras.layers.Dense(480, activation='relu')(x)
#x = tf.keras.layers.Dense(2048, activation='relu')(x)
x = tf.keras.layers.Dense(2048, activation='relu')(x)
x = tf.keras.layers.BatchNormalization()(x)
x = tf.keras.layers.Dropout(0.2)(x)

#x = tf.keras.layers.Dense(240, activation='relu')(x)
#x = tf.keras.layers.Dense(480, activation='relu')(x)
x = tf.keras.layers.Dense(2048, activation='relu')(x)
x = tf.keras.layers.BatchNormalization()(x)
x = tf.keras.layers.Dropout(0.2)(x)

#x = tf.keras.layers.Dense(32, activation='relu')(x)
#x = tf.keras.layers.Dense(64, activation='relu')(x)
#x = tf.keras.layers.Dense(128, activation='relu')(x)
x = tf.keras.layers.Dense(256, activation='relu')(x)
x = tf.keras.layers.BatchNormalization()(x)
# x = tf.keras.layers.Dropout(0.4)(x)

#outputs = tf.keras.layers.Dense(50, activation='softmax')(x)
#outputs = tf.keras.layers.Dense(100, activation='softmax')(x)
#outputs = tf.keras.layers.Dense(150, activation='softmax')(x)
outputs = tf.keras.layers.Dense(200, activation='softmax')(x)
model = tf.keras.Model(inputs, outputs)
```

+ Código + Texto

para essas arquiteturas essas foram as mtrizes de confusão



Matriz de confusão da Resnet

E por algum erro de estruturação não conseguimos plotar o gráfico da DenseNet

As precisões foram 0,3% e 28% respectivamente,

Para as arquiteturas com o transfer learning definidos dessa forma

Carregamento da ResNet50 pré-treinada: A linha base_model = tf.keras.applications.ResNet50(include_top=False, weights="imagenet", input_shape=(IMG_SIZE, IMG_SIZE, 3)) carrega a arquitetura da ResNet50 pré-treinada no conjunto de dados ImageNet. Essa rede neural já possui pesos aprendidos ao ser treinada em milhões de imagens de diferentes categorias.

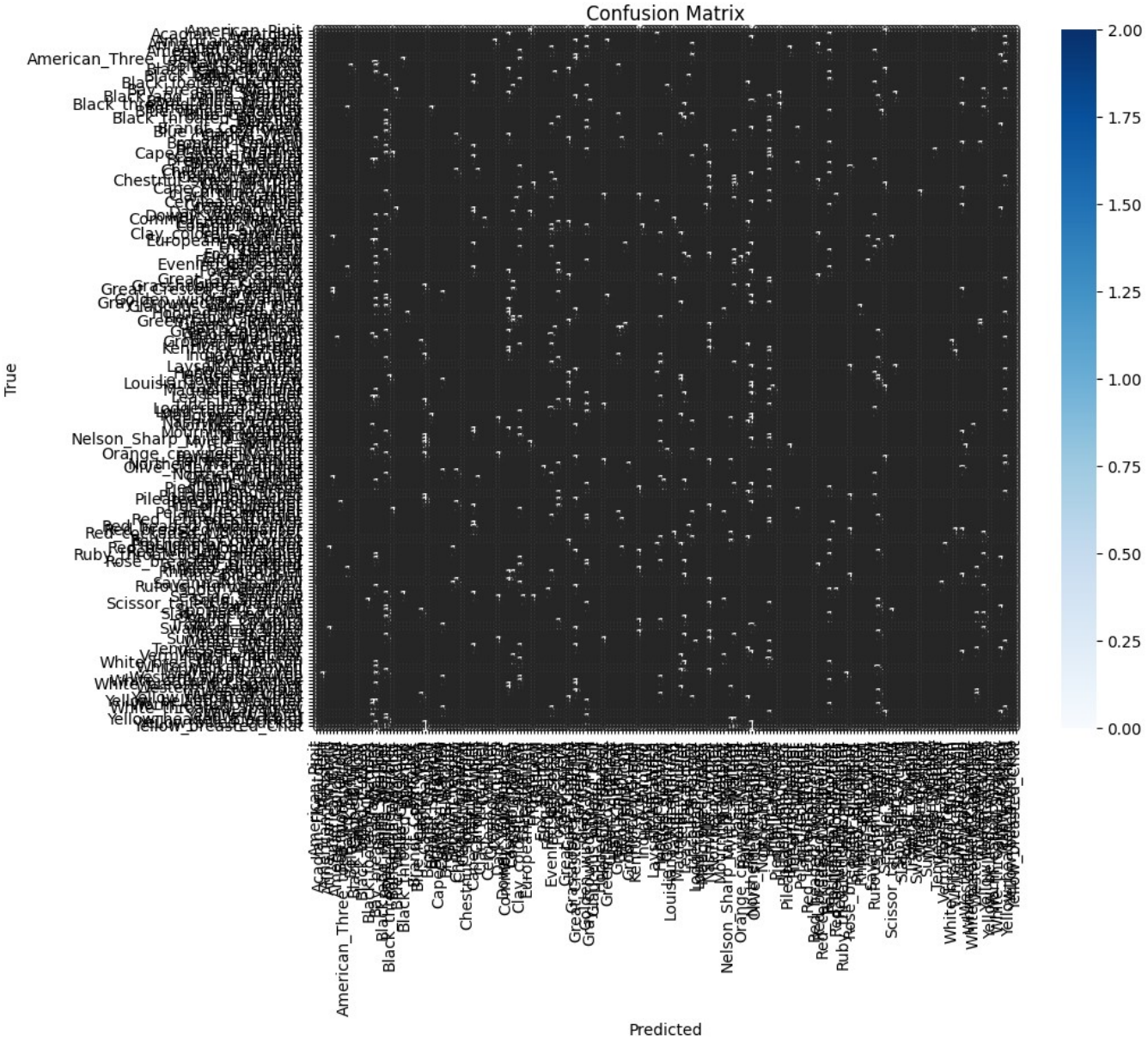
Congelamento dos pesos: Ao definir base_model.trainable = False, os pesos da ResNet50 são congelados, o que significa que eles não serão atualizados durante o treinamento posterior. Essa etapa é realizada para manter o conhecimento prévio da rede intacto e evitar que seja alterado.

Construção do modelo completo: O código utiliza tf.keras.Sequential() para construir o modelo completo. A base_model (ResNet50) é adicionada como a primeira camada, seguida por uma camada de GlobalAveragePooling2D(). Essa camada reduz a dimensionalidade da saída da ResNet50, agregando as informações mais relevantes de cada mapa de características em um único vetor.

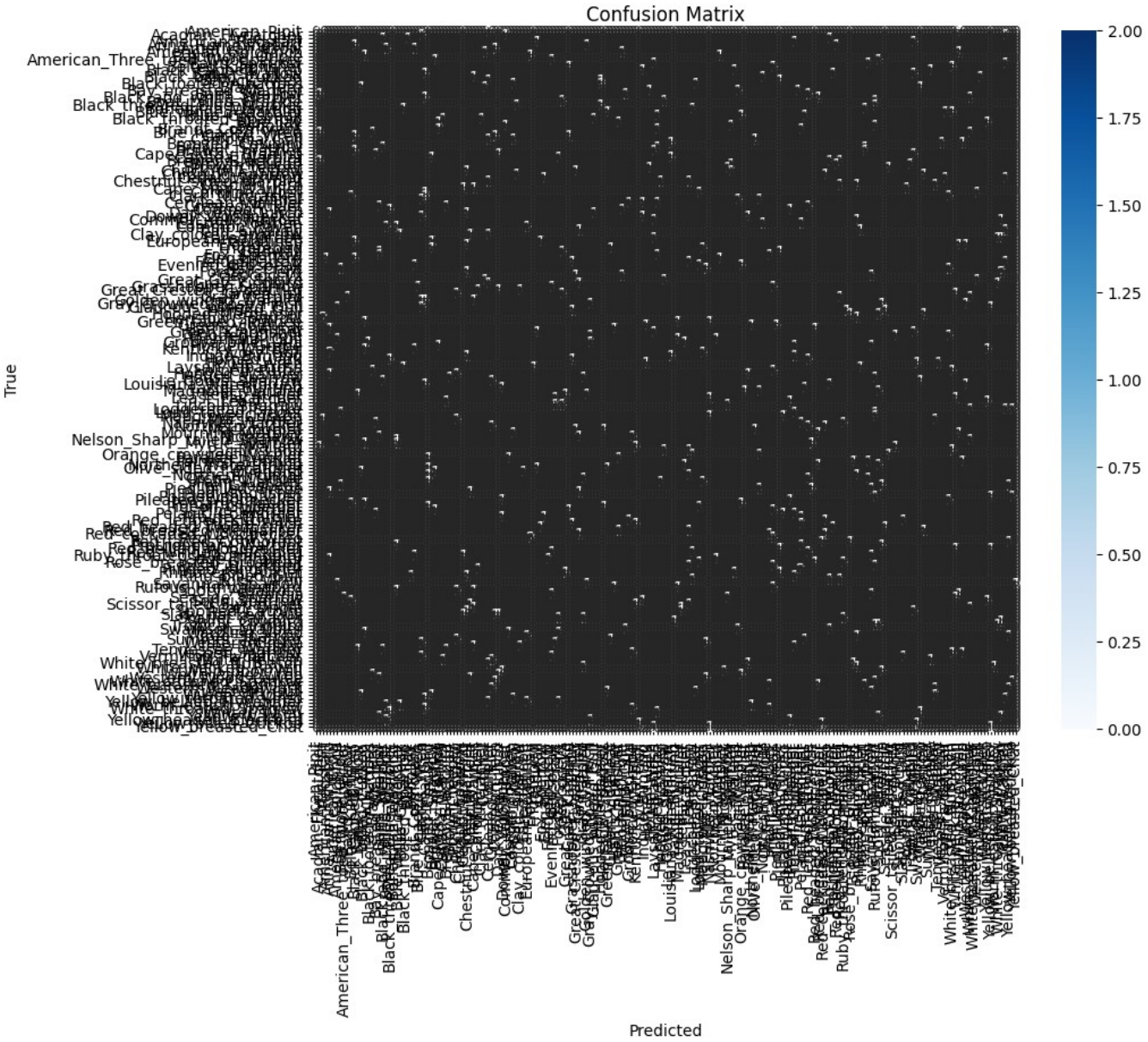
Camada de classificação: Uma camada totalmente conectada (Dense) com 200 neurônios é adicionada ao final do modelo. Essa camada utiliza uma função de ativação softmax para produzir uma distribuição de probabilidade sobre as 200 classes de saída possíveis.

```
base_model = tf.keras.applications.ResNet50(include_top=False, weights="imagenet", input_shape=(IMG_SIZE, IMG_SIZE, 3))
base_model.trainable = False

model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(200, activation='softmax')
])

model.summary()
```

,essas são as matrizes de confusão



Matriz de confusão da resnet



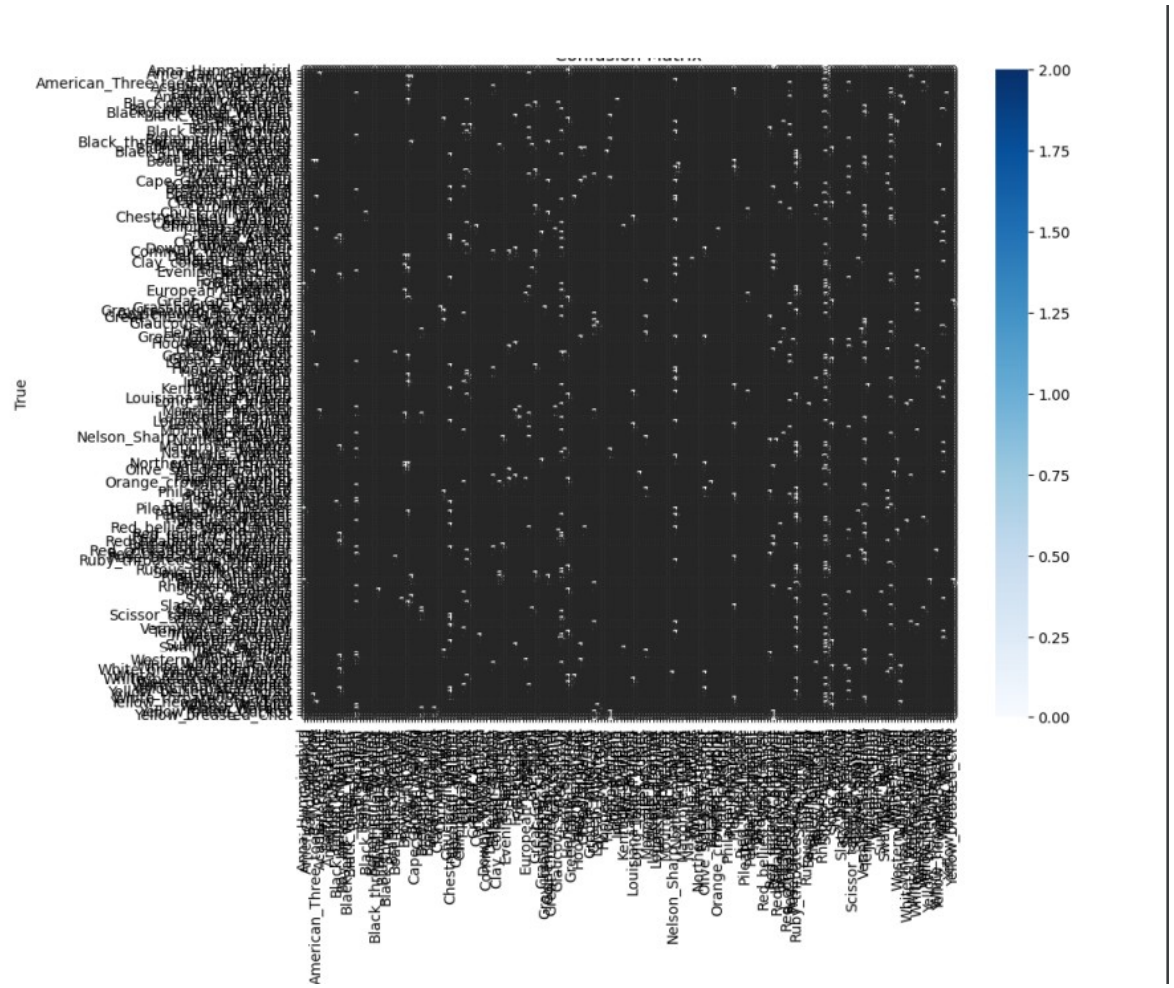
Matriz de confusão da Densenet
As acurácias obtidas foram de 1% e 94% respectivamente.

Agora para a terceira e última alteração no trasnfer learning das arquiteturas

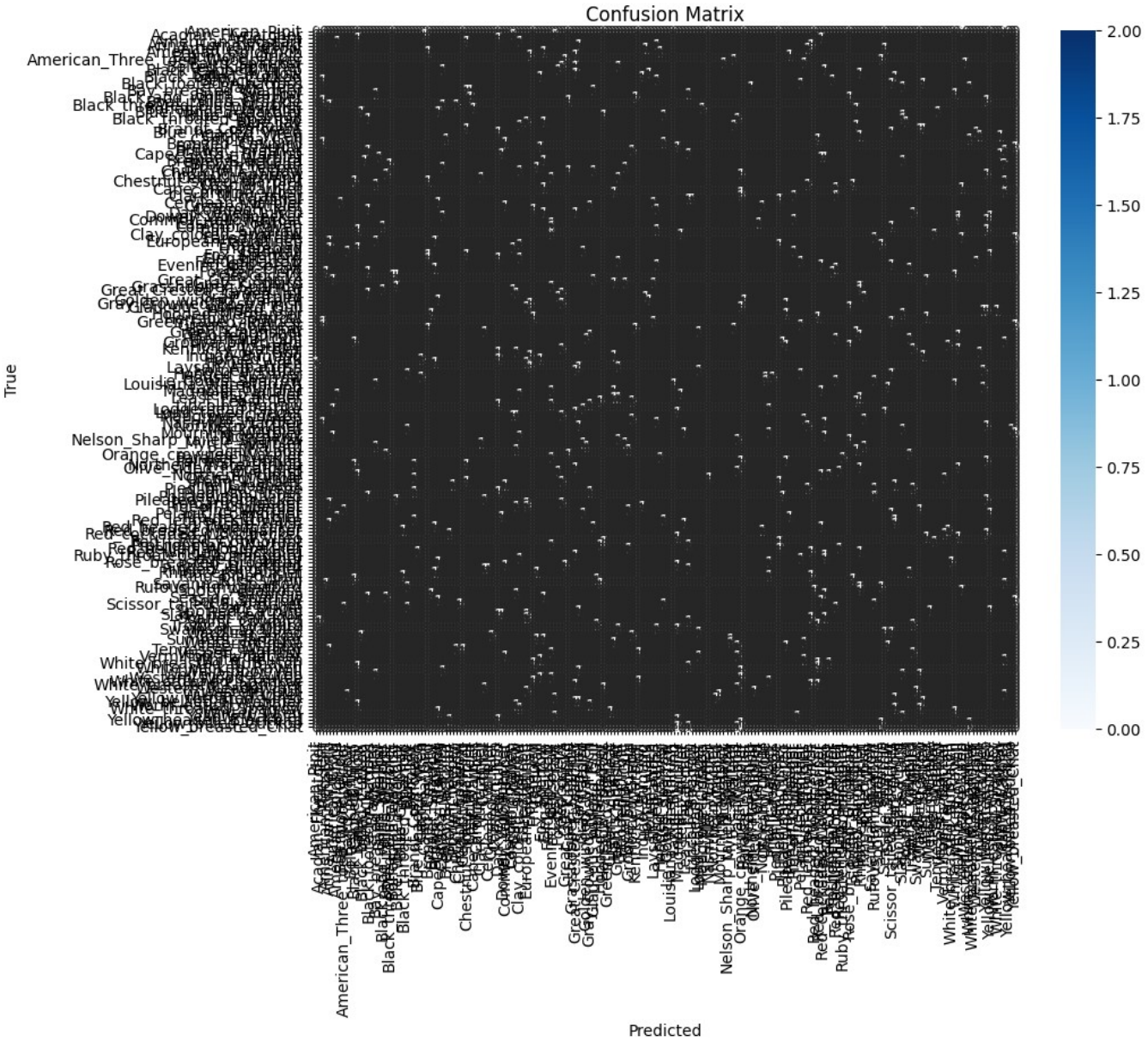
```
IMG_SHAPE = (IMG_SIZE, IMG_SIZE) + (3,)
base_model = DenseNet121(include_top=False, weights="imagenet", input_shape=IMG_SHAPE)
base_model.summary()

base_model.trainable = False
inputs = tf.keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
x = base_model(inputs, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)

x = tf.keras.layers.Dense(1024, activation='relu')(x)
x = tf.keras.layers.BatchNormalization()(x)
x = tf.keras.layers.Dropout(0.5)(x)

x = tf.keras.layers.Dense(512, activation='relu')(x)
x = tf.keras.layers.BatchNormalization()(x)
x = tf.keras.layers.Dropout(0.5)(x)

outputs = tf.keras.layers.Dense(len(class_names), activation='softmax')(x)
model = tf.keras.Model(inputs, outputs)
model.summary()
```

Esse foi o código desenvolvido para o treinamento onde,

IMG_SHAPE = (IMG_SIZE, IMG_SIZE) + (3,) define o formato da imagem de entrada, onde IMG_SIZE é o tamanho desejado para a imagem e (3,) indica que a imagem tem 3 canais de cor (RGB).
base_model = DenseNet121(include_top=False, weights="imagenet", input_shape=IMG_SHAPE) carrega a arquitetura da DenseNet121 pré-treinada no conjunto de dados ImageNet. A opção include_top=False indica que a camada de classificação original da DenseNet não será incluída, pois será substituída por uma nova camada de classificação personalizada para a tarefa específica.
base_model.summary() exibe um resumo da arquitetura do modelo DenseNet121 pré-treinado, mostrando as camadas e os parâmetros treináveis em cada uma delas, bem como o número total de parâmetros do modelo.
base_model.trainable = False congela os pesos da DenseNet121 pré-treinada para que não sejam atualizados durante o treinamento posterior.
inputs = tf.keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3)) define a camada de entrada do modelo com o formato especificado.
x = base_model(inputs, training=False) aplica a DenseNet121 pré-treinada às entradas para obter as saídas correspondentes.
x = tf.keras.layers.GlobalAveragePooling2D()(x) adiciona uma camada de pooling global média para reduzir a dimensionalidade das saídas.
Em seguida, há duas camadas Dense com função de ativação relu seguidas por camadas de BatchNormalization e Dropout. Essas camadas são adicionadas para adicionar não-linearidade, normalização dos dados e regularização para evitar overfitting durante o treinamento.
A última camada Dense possui um número de neurônios igual ao número de classes (len(class_names)) e usa a função de ativação softmax para gerar a distribuição de probabilidade sobre as classes.
Finalmente, o modelo é construído usando tf.keras.Model(inputs, outputs) onde inputs é a camada de entrada e outputs é a camada de saída.
model.summary() exibe um resumo da arquitetura completa do modelo, mostrando as camadas e os parâmetros treináveis em cada uma delas, bem como o número total de parâmetros do modelo.



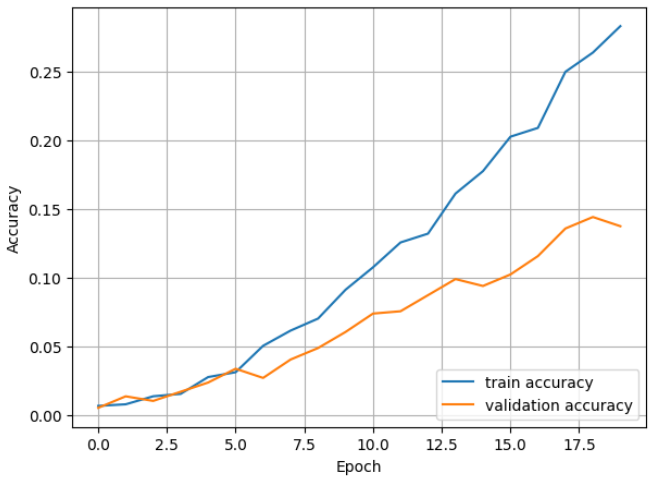
Matriz de confusão da resnet

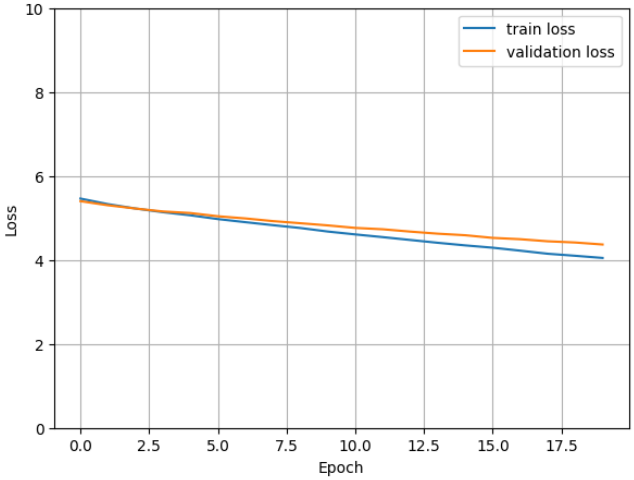


Matriz de confusão da densenet
As acurácias foram 0,1% e 32% respectivamente.

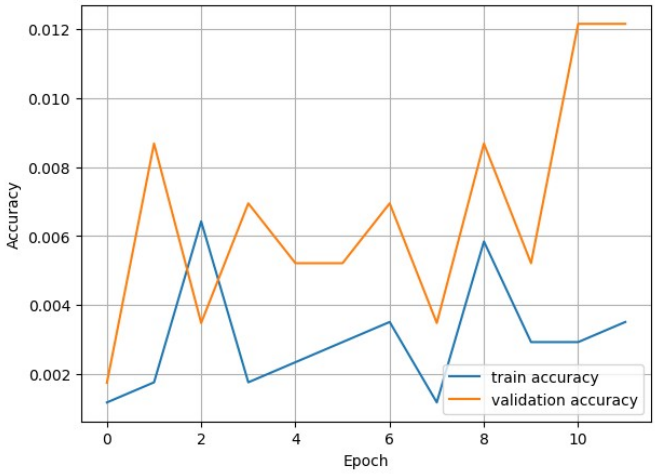
Devido a problemas na compilação dos códigos não conseguimos plotar algumas das evolução das épocas de cada uma das redes

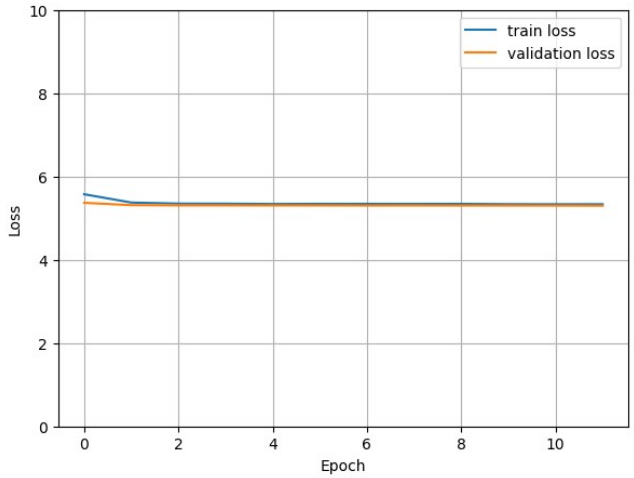
Resnet sem TL, não conseguimos gerar o gráfico



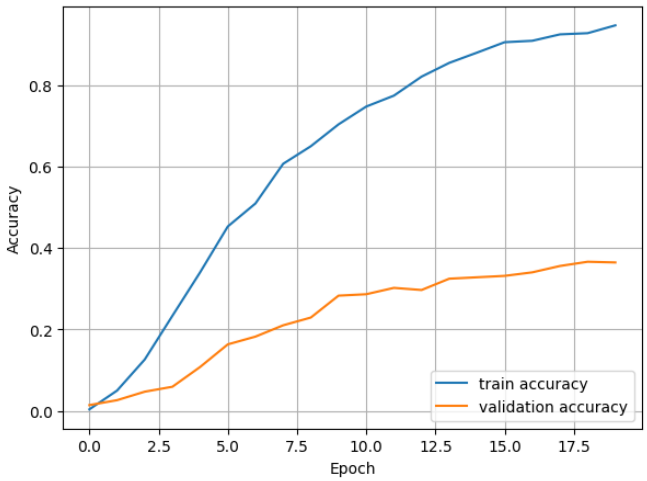


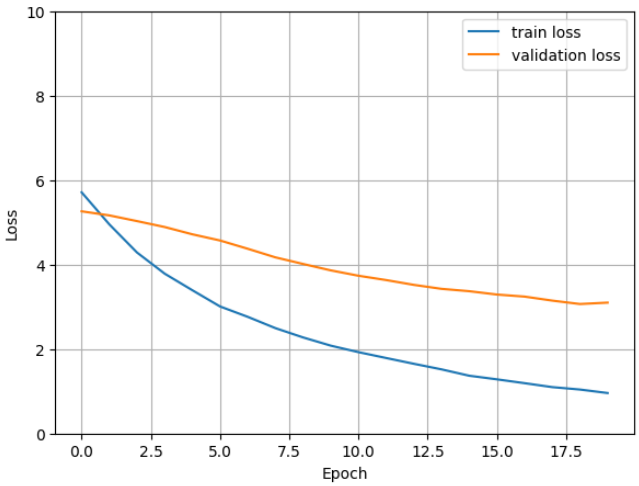
Densenet sem TL.



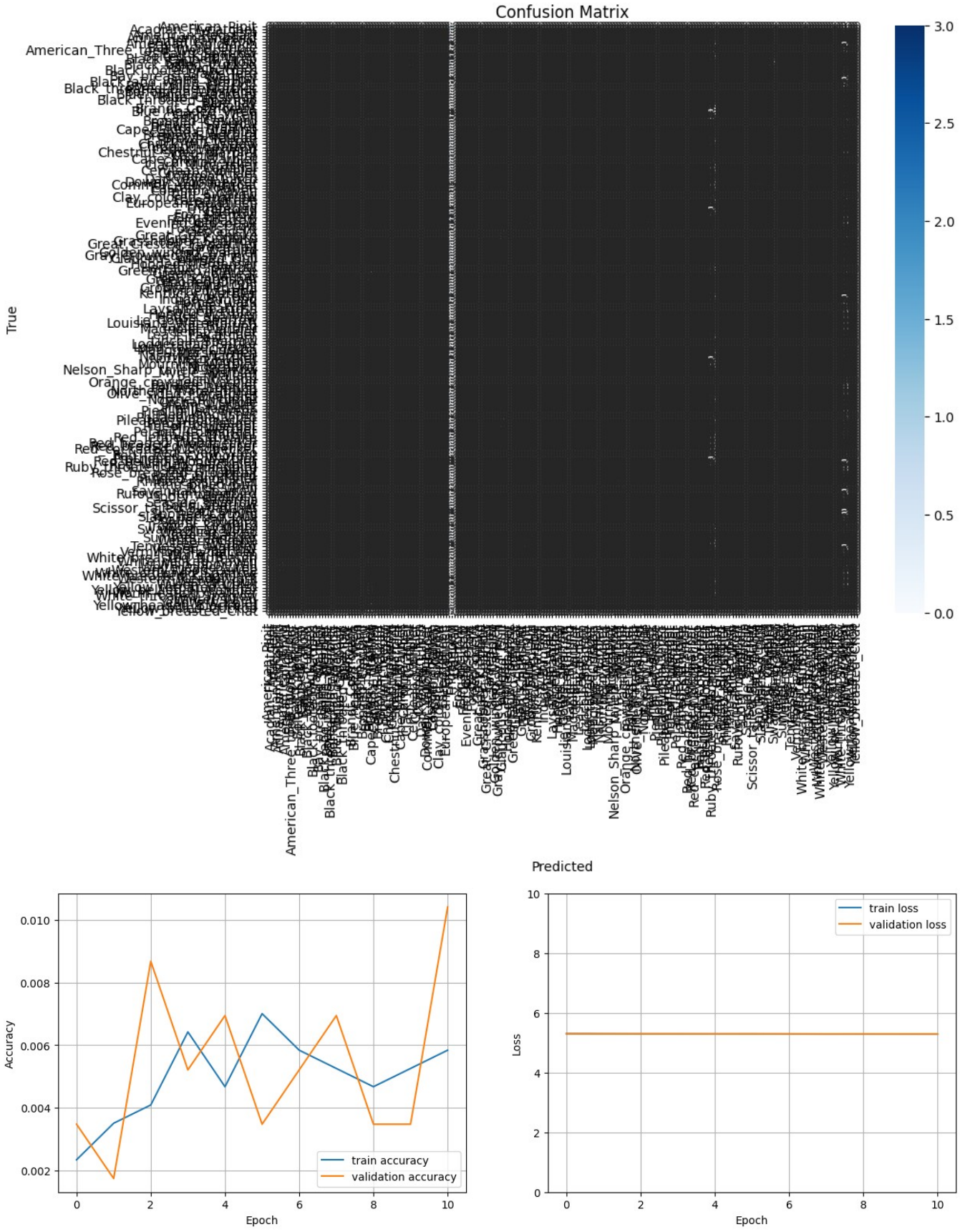


Resnet primeiro TL.





DenseNet primeiro TL.
ResNet segundo TL, não conseguimos gerar o gráfico.
DenseNet segundo TL, não conseguimos gerar o gráfico.
Por fim a nossa CNN, essa foi a que menos performou



Isso se da pelo pouco treinamento e poucos dados, como as outras redes usam transfer learning elas já tem pesos salvos e podem recorrer a eles creio que com uma maior quantidade de dados e um maior treinamento, chegaríamos a um melhor modelo.

VI. Conclusão

Podemos concluir que a melhor das arquiteturas utilizadas foi a DenseNet com a primeira configuração de transfer learning, que além de ter a maior das acurácias foi a que teve menor perda.