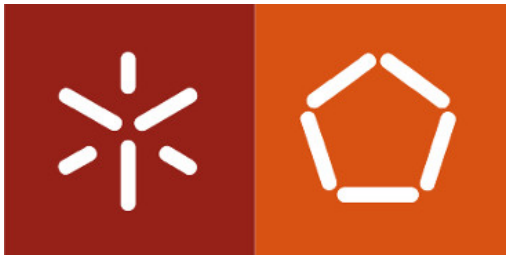

Universidade do Minho

Mestrado em Engenharia Informática

Exploração de modelos de *Deep Learning*

Diogo Tavares, PG42826

Pedro Santos, PG42847

Valeriy Apostolyuk, PG42647

Tecnologias e Aplicações

4º Ano, 2º Semestre

Departamento de Informática

Junho 2021

Índice

Lista de Figuras

1 Introdução

1.1 Identificação do Projeto e Objetivos

No âmbito da Unidade Curricular de Tecnologias e Aplicações foi-nos proposta a realização de um modelo de *Deep Learning*, capaz de identificar os tipos de sinais de trânsito alemães.

O principal objetivo passa pela criação de um modelo que consiga identificar qual o sinal de trânsito que está presente numa determinada imagem.

1.2 Etapas de Trabalho

Este trabalho foi dividido em várias etapas, sendo elas, a obtenção e tratamento do *dataset* de imagens, a criação do modelo e a aplicação de várias técnicas de *deep learning*, que permitam facilitar o treino e avaliação do modelo. Essas técnicas consistem em *data augmentation*, *ensembles* e até *transfered learning*.

1.3 Estrutura do relatório

Este relatório encontra-se dividido em 6 capítulos.

No capítulo 1.**Introdução** é feita uma breve abordagem ao que se pretende com a realização deste projeto, bem como quais as diferentes etapas do trabalho.

Seguidamente, no capítulo de 2.**Estado da Arte** faz-se uma explicação de alguns métodos existentes, que permitam realizar estas tarefas, bem como melhorar a *performance* do modelo.

No capítulo 3.***Dataset***, faz-se uma explicação de como é constituido o *dataset*.

No capítulo 4.**Modelo** é descrito o modelo criado.

No capitulo 5.**Resultados** fazemos a reflexão e análise dos resultados obtido com a aplicação do modelo criado.

Por fim, em 6.**Conclusão** é feita uma breve conclusão do trabalho, onde são retiradas ilações sobre o mesmo e se foram concluídos os objetivos propostos.

2 Estado da Arte

2.1 Deep Learning

O *deep learning* faz parte de uma família de métodos de *machine learning* baseado nas redes neuronais. A aprendizagem pode ser supervisionada, semi-supervisionada ou não supervisionada. As arquiteturas do *deep learning*, como as Redes Neuronais Profundas, de Gráfico e Redes Neuronais Convolucionais (CNN), foram aplicadas em campos como a Visão Computacional, Reconhecimento de Fala, Processamento de Linguagem Natural, Análise de Imagens relacionadas com a Medicina, Programas de Jogos de Tabuleiro, entre outros. As Redes Neuronais Artificiais (RNA), foram inspiradas no processamento de informações e nos nós de comunicação distribuída em sistemas biológicos. Estes sistemas melhoram progressivamente as suas capacidades de realizar tarefas. No reconhecimento da imagem, as RNAs podem aprender a identificar imagens que contêm um certo animal (ex: gato), ao analisar imagens que foram marcadas manualmente como "é gato", ou "não é gato"e utilizando os resultados analíticos para identificar gatos de outras imagens. A maioria dos modelos do *deep learning* moderno são baseados especialmente em CNN's. Uma CNN convolve recursos aprendidos, com dados de input e usa camadas convolucionais 2D, logo é ideal para o processamento de imagens de duas dimensões. Em comparação com os outros algoritmos de classificação de imagem, as CNN podem aprender os filtros que precisam de ser feitos à mão em outros algoritmos. As CNN têm 2 camadas: *input* e *output*; e várias camadas ocultas "*hidden*". Essas camadas geralmente consistem em camadas convolucionais, *ReLU* (Unidade Linear Retificada), agrupamento e camadas totalmente conectadas. As camadas convolucionais aplicam uma operação de convolução no input, no qual passa as informações para a próxima camada, as camadas de agrupamento combinam os *outputs* de *clusters* de neurónios num único neurónio na próxima camada e as camadas conectadas ligam todos os neurónios numa única camada. Resumindo, uma CNN:

- Começa com uma imagem de input;
- Aplica vários tipos de filtros para construir um mapa;

- Aplica uma função $ReLU$ para incrementar a não linearidade;
- Aplica uma camada de agrupamento para cada mapa;
- Achata as imagens agrupadas num vetor longo;
- Insere o vetor numa rede neural artificial totalmente conectada;
- Processa os recursos pela rede. A camada final fornece a "votação" das classes que localizamos;
- Treina através da propagação direta e retropropagação por muitas, muitas épocas. Isso repete-se até que haja uma rede neural bem definida com pesos treinados e detectores de recursos.

2.2 Data Augmentation

Ter um grande conjunto de dados é crucial para o desempenho do modelo de *deep learning*, no entanto, podemos melhorar o desempenho do modelo ao ampliar os dados existentes. O *data augmentation*, na análise de dados, é uma técnica utilizada para aumentar a quantidade de dados, adicionando cópias ligeiramente modificadas de dados já existentes, ou dados sintéticos recém-criados a partir de dados existentes. Esta técnica, atua como um regularizador e ajuda a reduzir o *overfitting* ao treinar um modelo de *machine learning*. No *data augmentation*, existem vários pacotes:

1. ***imgaug***: é uma biblioteca para ampliação da imagem em experiências de *machine learning*. Suporta várias técnicas de ampliação de imagem, permite combiná-las facilmente e executá-las em ordem aleatória ou em vários cores de um CPU. Isto torna muito fácil de aplicar o pacote ao conjunto de dados que contém imagens para problemas de segmentação e detecção de objetos.
2. ***Albumentations***: é escrito com base no *numpy*, *OpenCV* e *imgaug*. Este pacote é capaz de mais de 60 transformações de nível de pixel e nível espacial, transformar imagens com máscaras, caixas delimitadoras e pontos-chave, organizar incrementos em *pipelines* ou integração com o *PyTorch*.

3. ***Augmentor***: contém menos incrementos possíveis do que os pacotes anteriores, mas tem excelentes recursos como rotações para preservação de tamanho, corte com preservação de tamanho e corte, que é mais adequado para o *machine learning*. O pacote também permite compor *pipelines* de incremento e usá-los com o *PyTorch*.

2.3 Ensemble

Em estatística e *machine learning*, os métodos de *ensemble* utilizam vários algoritmos de aprendizagem para obter melhor desempenho preditivo. Ao contrário de um *ensemble* estatístico, que geralmente em mecânica estatística é infinito, um *machine learning ensemble* consiste em apenas um conjunto finito concreto de modelos alternativos, mas normalmente permite que uma estrutura muito mais flexível exista entre essas alternativas. Supostamente, os *ensembles* tendem a render melhores resultados quando existe uma diversidade significativa entre os modelos. Muitos métodos procuram promover a diversidade entre os modelos que combinam. Relativamente aos tipos de *Ensemble* existentes temos:

1. **Classificador ótimo de *Bayes***: é uma versão que assume que os dados são condicionalmente independentes da classe e torna o cálculo mais viável. Cada hipótese, recebe um voto proporcional à probabilidade de que o conjunto de dados de treino seria amostrado de um sistema, se essa hipótese fosse verdadeira. Para facilitar o treino de dados de um conjunto finito, o voto de cada hipótese também é multiplicado pela probabilidade anterior dessa hipótese.

2. **Agregação de *Bootstrap***: também conhecido como *bagging*, envolve ter cada modelo no voto do conjunto com um peso equivalente. Este método, treina cada modelo usando um subconjunto desenhado aleatoriamente do conjunto de treino. Na Agregação de *bootstrap*, as amostras são geradas de forma que estas sejam diferentes umas das outras, porém a substituição é permitida. Substituição significa que uma instância pode ocorrer em várias amostras várias vezes, ou pode não aparecer em algumas amostras. Essas amostras são então fornecidas a várias pessoas e, em seguida, os resultados

de cada pessoa são combinados na forma de votação.

3. ***Boosting***: envolve a construção incremental de um *ensemble* ao treinar cada nova instância de modelo, para enfatizar as instâncias de treino que os modelos anteriores classificaram incorretamente. Em alguns casos, o *boosting* demonstrou ter melhor precisão do que o *bagging*, mas também tende a ser mais provável de ajustar os dados de treino.

4. ***Stacking***: envolve treinar um algoritmo de aprendizagem, para combinar as previsões de vários outros algoritmos. Primeiro, todos os outros algoritmos são treinados usando os dados disponíveis, depois um algoritmo de combinação é treinado para fazer uma previsão final, usando todas as previsões dos outros algoritmos como *inputs* adicionais. Se um algoritmo de combinação arbitrário for utilizado, o empilhamento pode teoricamente representar qualquer uma das técnicas de combinação, embora, na prática, um modelo de regressão logística seja frequentemente utilizado como combinador.

2.4 Transfered Learning

O *transfered learning* é um problema de pesquisa no *machine learning*, que se concentra em armazenar o conhecimento adquirido ao resolver um determinado problema e aplicá-lo a um outro, que seja relacionado. Atualmente, é muito popular no *deep learning*, dado que pode treinar redes neuronais profundas com comparativamente poucos dados. A ideia geral é usar o conhecimento que um modelo aprendeu, com uma tarefa com muitos dados de treino rotulados, disponíveis para uma nova tarefa que não possui muitos dados. Este método é muito utilizado em tarefas de visão computacional e processamento de linguagem natural, e apesar de não ser considerado como uma técnica do *machine learning*, pode ser visto como uma metodologia de *design* dentro do campo, por exemplo, *active learning*.

3 *Dataset*

O *dataset* que foi utilizado para treinar e avaliar a *preformance* do modelo criado, foi o *dataset GTSRB*, contendo imagens de sinais de trânsito alemães. Este consiste em dois *datasets* diferentes, um de treino com 39210 imagens, onde estas estão divididas em 43 classes diferentes de sinais de trânsito, e outro de teste com 12630 imagens. O *dataset* de treino, foi dividido num *dataset* de treino e num de validação, tendo sido esta divisão feita com um rácio de 70% para treino e os restantes 30% para validação, resultando assim em 27930 imagens de treino e 11280 de validação.

Na figura 1, podemos observar um conjunto de imagens exemplificativas das que se encontram presentes no *dataset* de treino.

Figura 1: Imagens exemplificativas do *Dataset* de treino

4 Modelo

O modelo que foi utilizado para treinar e avaliar o *dataset*, consiste modelo convolucional sequencial, com várias camadas de forma a fazer o processamento dos dados.

Este modelo é composto por várias camadas de processamento, que são:

- *Layer* inicial convolucional, *Conv2D*, com 128 filtros, 5 por 5, uma função de ativação *LeakyReLU*, um *batch normalization* e um *dropout* a 0.5
- *Layer* convolucional, *Conv2D*, com 196 filtros, 5 por 5, uma função *max pooling*, com *pool_size* de 2 por 2, uma função de ativação *LeakyReLU*, um *batch normalization* e um *dropout* a 0.5
- *Layer* convolucional, *Conv2D*, com 256 filtros, 5 por 5, uma função *max pooling*, com *pool_size* de 2 por 2, uma função de ativação *LeakyReLU*, um *batch normalization* e um *dropout* a 0.5
- *Layer* que contêm um *Flatten*, um Dense com valor de output 384, uma função de ativação *LeakyReLU* e um *dropout* a 0.5
- *Layer* final, sendo uma camada densa para dar um output com o número de classes que se pretende

5 Implementação e Resultados

5.1 Técnicas de *data augmentation* aplicadas

De modo a melhorar os resultados, foram aplicadas técnicas de *data augmentation* para aumentar a diversidade do *dataset* de treino.

- **Luminosidade**

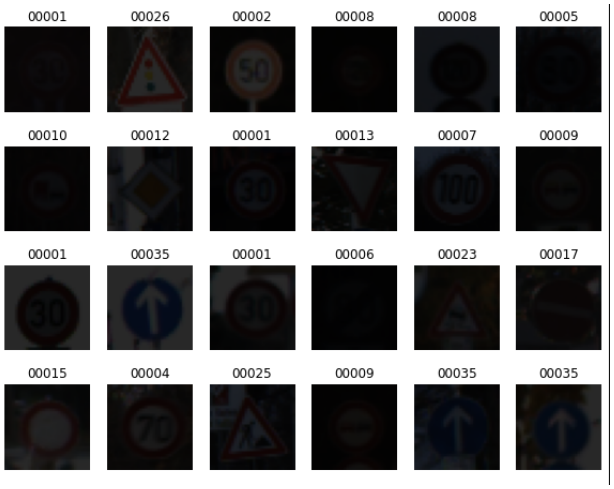

Figura 2: *Dataset* processado com luminosidade

- **Contraste**

Figura 3: *Dataset* processado com contraste

- ***HUE***

Figura 4: *Dataset* processado com *hue*

- **Saturação**

Figura 5: *Dataset* processado com saturação

- **Rotação**

Figura 6: *Dataset* processado com rotações

- ***Shear***

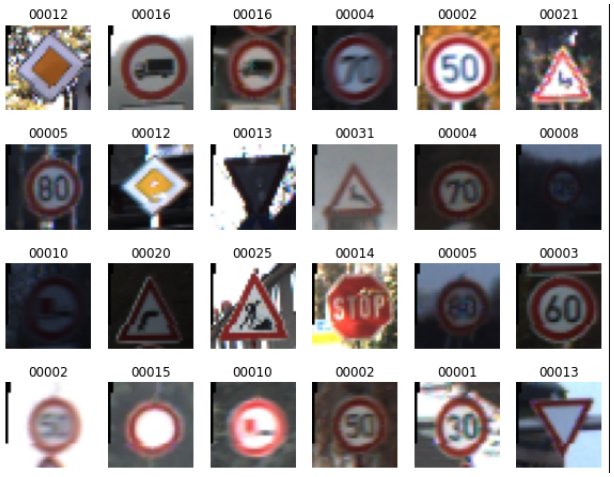

Figura 7: *Dataset* processado com *shear*

- **Translação**

Figura 8: *Dataset* processado com translações

- ***Perlin noise*** - O *perlin noise* é um tipo de ruído gradiente, usado na área da computação gráfica para aumentar o aspeto realista da imagem.

Figura 9: *Dataset* processado com *Perlin noise*

- ***Gaussian noise*** - O ruído gaussiano, é um ruído estatístico cuja função de probabilidade é igual à da distribuição normal, que é também conhecida como distribuição gaussiana.

Figura 10: *Dataset* processado com *Gaussian noise*

5.2 Treinos e testes realizados

Usando diferentes variações das funções de *data augmentation* descritas, realizamos diferentes treinos de redes.

Cada treino consistiu em correr 3 vezes o modelo e 100 épocas em cada, onde os resultados foram agrupados, produzindo assim um resultado médio para cada configuração de *data augmentation*. Uma das versões, no entanto, consistiu em correr 9 vezes o modelo e 50 épocas cada. De notar que, apesar de definido um número de 100 épocas nenhum dos treinos não passou das 40 épocas, devido ao critério de paragem (*early stopping*).

- **V0** - Nesta versão foram introduzidas imagens de todas as funções descritas anteriormente excepto as funções de ruído resultando num total de 223 440 imagens no conjunto de treino. Foram treinados 9 modelos com 50 épocas cada.
- **V1** - Versão idêntica à anterior mas com apenas 3 modelos e 100 épocas cada.

- **V3** - A esta versão foi adicionado o *dataset* inicial sem qualquer tipo de processamento resultando em 251 370 imagens de treino.

  Uma vez que a introdução do *dataset* inicial sem qualquer tipo de processamento não apresentou resultados melhores as restantes versões foram baseadas na versão **V1**.

- **V4** - Baseado na versão **V1**, a esta versão foi adicionada a função de *perlin noise* com um parâmetro de frequência igual a 1024.

- **V5** - Neste treino, o parâmetro de frequência para o *perlin noise* foi alterado para 512.

- **V6** - Parâmetro de frequência do *perlin noise* alterado para 2048.

- **V7** - Nesta versão foram usadas as funções da versão **V1** em conjunto com o *gaussian noise*, com um desvio padrão de $25/255 = 0.09803921568$.

- **V8** - Idêntico à versão**V7**, com um desvio padrão de $50/255 = 0.19607843137$.

- **V9** - Valor de desvio padrão alterado para $75/255 = 0.29411764705$.

- **V10** - Com base na **V1**, foram adicionadas imagens processadas por *perlin noise* com uma frequência de 1024 e imagens processadas pelo *gaussian noise* com valor de desvio padrão igual a 50/255, resultando em 279 300 imagens de treino.

- **V11** - A frequência do *perlin noise* foi mantida, alterando o valor de desvio padrão para 25/255.

- **V12** - A frequência do *perlin noise* foi alterada para 512, mantendo o valor de desvio padrão igual a 25/255.

  Nota: A versão 2 foi descartada devido a um erro de implementação

Os resultados dos treinos realizados podem ser observados nas figuras 11, 12 e 13.

VERSION	ACCURACY MÉDIA	LOGITS ACCURACY
0	99,336	0,994853523
1	99,364	0,995249406
3	99,266	0,993824228
4	99,398	0,995011876
5	99,419	0,994615994
6	99,303	0,994140934
7	99,303	0,994378464
8	99,134	0,992794933
9	99,084	0,992794933
10	99,111	0,993032462
11	99,232	0,993586698
12	99,235	0,993982581

Figura 11: Resultados dos treinos

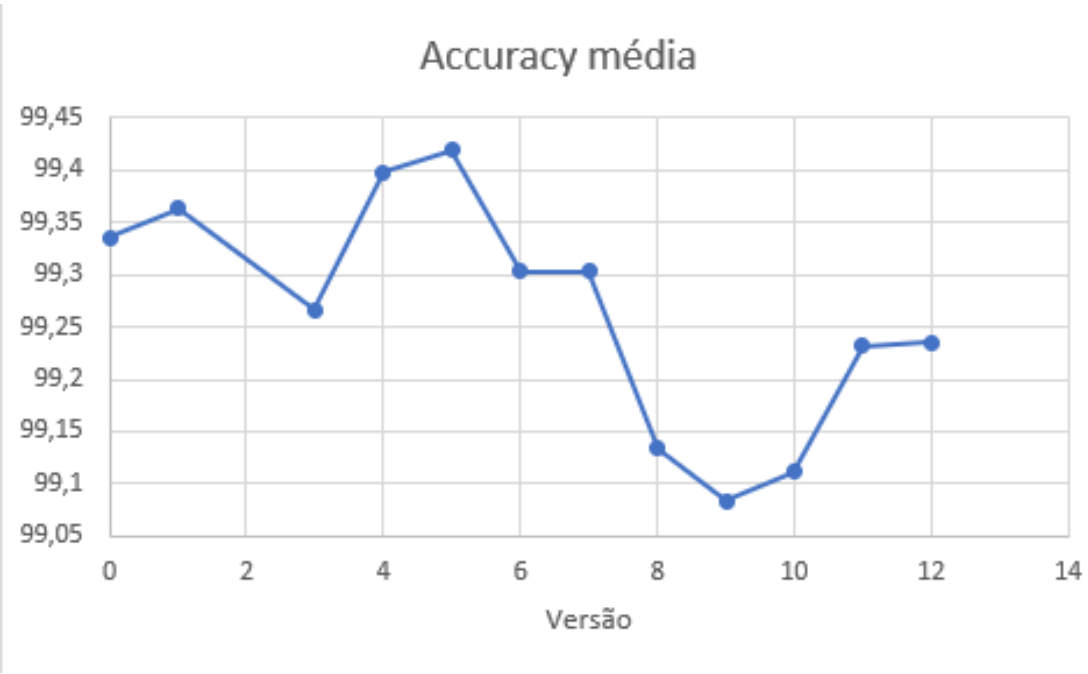

Figura 12: *Accuracy* média em função da versão

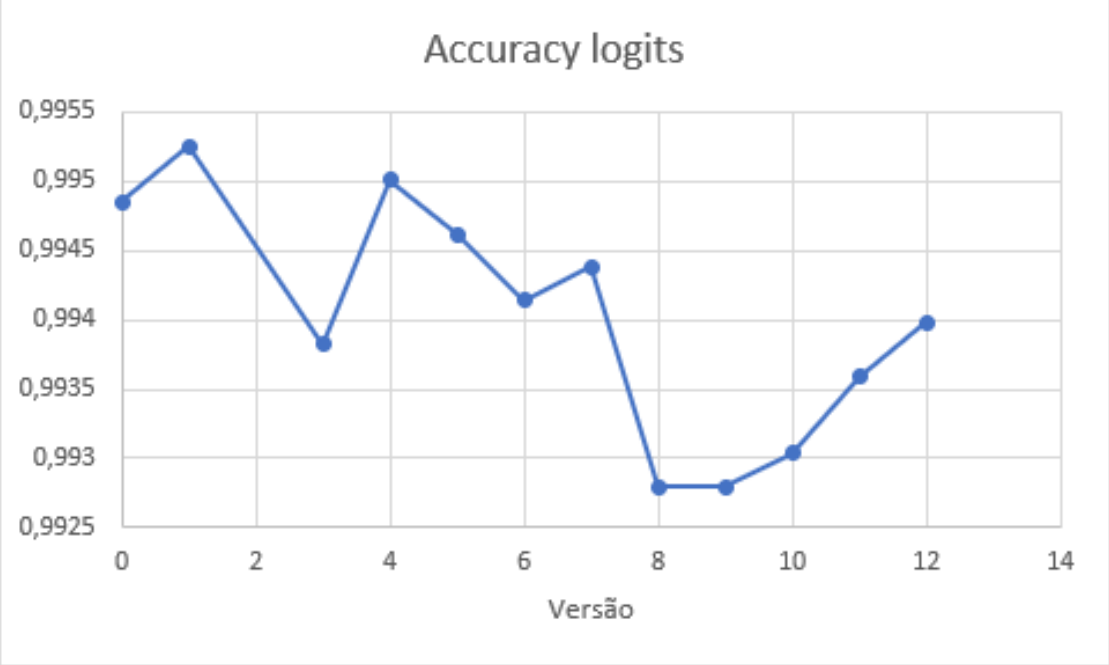

Figura 13: *Accuracy* dos *logits* em função da versão

Analisando os resultados obtidos podemos concluir que as versões 4 e 5, que implementaram processamento com *perlin noise*, foram as que apresentaram melhores resultados em termos de *accuracy* média de entre os 3 modelos treinadas com o mesmo *dataset*.

Do ponto de vista dos resultados apresentados pelos *logits*, estes foram melhores nas versões 1 e 4.

6 Conclusão

Com o objetivo de conseguir identificar sinais de trânsito alemães, foi criado um modelo sequencial, com várias camadas convolucionais e densas. Para realizar o treino deste modelo, foi utilizado um *dataset* de treino, com 39210 imagens, que foi dividido em treino e validação, tendo no final ficado 27930 imagens de treino e 11280 de validação. Para testar o mesmo, foi utilizado um *dataset* com 12630 imagens.

Através deste modelo e recorrendo a técnicas de *deep learning*, como *ensembles* e *data augmentation*, foram treinadas e avaliadas várias redes, de modo a se conseguir perceber quais as combinações que resultam em melhores resultados.

Como tal, no que toca a técnicas de *data augmentation* que foram aplicadas, temos variações na luminosidade, contraste, *HUE*, saturação, rotação, *shear*, translação, *perlin noise* ou mesmo *gaussian noise*.

Através dos vários testes que foram realizados, percebemos que no geral todos os resultados foram relativamente próximos uns dos outros, mas as duas versões que mais se destacam no meio dos resultados são a versão 4 e 5. Estas versões conseguiram resultados de 99.398 e 99.419, respetivamente, em relação à *accuracy* média e 99.5 e 99.46, respetivamente, em relação à *accuracy* dos *logits*.

Estas duas versões apenas variam uma da outra relativamente ao parâmetro de frequência, sendo que na versão 4 é de 1024 e na versão 5 é de 512.

Assim, achamos que foram cumpridos os objetivos do projeto, com o modelo e as alterações feitas ao mesmo.

7 Referências

Deep Learning:

Deep Learning - https://en.wikipedia.org/wiki/Deep_learning

Towards Data Science - https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb

Data Augmentation:

Data augmentation - https://en.wikipedia.org/wiki/Data_augmentation

Technopedia - https://www.techopedia.com/definition/28033/data-augmentation

Towards Data Science - https://towardsdatascience.com/data-augmentation-for-deep-learning-4fe21d1a4eb9

Ensemble:

Ensembles - https://en.wikipedia.org/wiki/Ensemble_learning

TechTalks - https://bdtechtalks.com/2020/11/12/what-is-ensemble-learning/

Transfered Learning:

Built In - https://builtin.com/data-science/transfer-learning